Explore

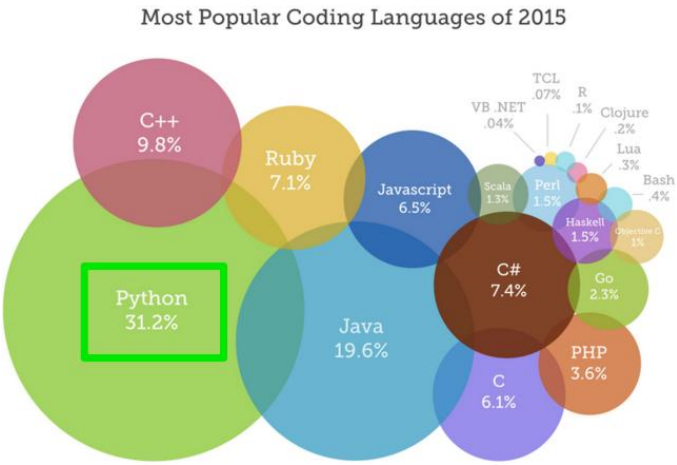
Learn Python by Doing It Guide

By David J. Bouwsma 2019 ©

Contents

Installing Python

How to get Python



Python is one of the world's most commonly used computer languages. It is open-source and completely free, too. Here are two ways to get Python:

WAY 1:

Run it on-line.Type trinket.io into a web browser and program on-line.https://trinket.io/python[1]

WAY 2:

Get Python for your computer. This is what we will do in class.

Run a browser and search for **Thonny**.[2] Download the installer, run it and customize your Python with goodies like pygame, numbpy, scipy, and more for a fuller experience. Then get **Zelle's graphics**. Follow the steps below.

1. Go to https://thonny.org/. (See QR.)
2. Click either the **Windows link** or the **Mac link** on the site, depending upon the type of computer you are using.
3. **Download** the installer.
4. **Run** the installer from the download folder.
5. To add **modules** do the following:
    a. **Run** Thonny.
    b. Click on **Tools** (at the top.)
    c. Click **Manage Packages**.
    d. **Type in** a module (package) name (just under the title bar.)
    e. Click **Find in pypl**.
    f. Click **Install**.
    g. **Repeat** steps d through f for all modules.

In this book we use:
pygame
Here are some handy modules:

Thonny Site

Zelle graphics site

[1] One of the best things about Python is that people all over the world have added packages named modules that give Python additional capabilities. Here is a link to the list of modules included in trinket.io: https://trinket.io/docs/python

[2] Thonny is an **IDE**, that is, an **I**ntegrated **D**evelopment **E**nvironment developed by the University of Tartu, Estonia. IDEs are programs that let programmers use their mouse and keyboard to edit and run their programs. Thonny is an educational IDE. It is easy for students to use, features a module installer, and contains its own internal version of Python 3, making it very easy for schools to manage, as well.

6. Now get the zelle graphics package we will use later in the book.
    a. Go to the University of Wartburg at http://mcsp.wartburg.edu/zelle/python/ to **download** professor Zelle's graphics module. It is named **graphics.py**.
    b. Click on the **zelle.py** link (just past the pictures of books.)
    c. **Click** in the text.
    d. Hold down the **CTRL** key and press the **A** key to highlight everything.
    e. Hold down the **CTRL** key and press the **C** key to copy the program.
    f. Run Thonny (if it is not running already.)
    g. Click the word **File** (in the top bar of the program.)
    h. Click **New**.
    i. Hold down the **CTRL** key and press the **V** key to paste the program in.
    j. Click the word **File** again,
    k. Click **Save As**.
    l. Type in **graphics**(all small letters.)
    m. Click **Save**.

Turtle Graphics

Lines

Python has a really easy graphical package called **turtle graphics**. To run it, simply type the following lines in:

```
import turtle
win = turtle.Screen()
t=turtle.Turtle()
```

Be careful with capital letters. In python a capital **A** is a different letter than a small **a**. Find a the ▶ symbol, and click it to run your program. A new blank window should open with a turtle or arrow at its center point.

Now add the following text after the last line, then run the program again:

```
forward(50)
```

The line you see on your screen is 50 pixels (the computer term for dots) long.

The turtle can go backward as well as forward. Try this program:

```
import turtle
```

```
win = turtle.Screen()
t=turtle.Turtle()

backward(100)
```

forward(length)	Draw lines
backward(length)	

Turning

You can make the turtle turn, so you can draw lines at angles. Try this program

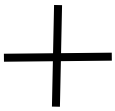
```
import turtle
win = turtle.Screen()
t=turtle.Turtle()

t.forward(50)
t.left(90)
t.forward(50)
t.left(90)
```

left(turn)	turn the turtle
right(turn)	

Plus Challenge:

Make a plus sign.



Color

The turtle can draw in different colors. Try out this example:

```
import turtle
win = turtle.Screen()
t=turtle.Turtle()

t.color("blue")
t.forward(50)
t.right(90)
t.forward(2)
t.right(90)

t.color("red")
t.forward(50)
t.right(90)
t.forward(2)
t.right(90)

t.color("green")
t.forward(50)
t.right(90)
t.forward(2)
```

color("color") sets color from now on

```
t.right(90)
```

Python colors include all the common colors, like:

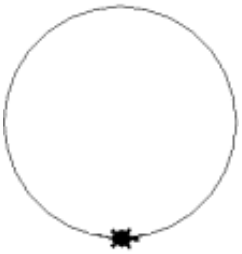
red,blue, green, orange,gray,yellow, purple,brown,black,plus many others.

Click here to see a chart of Python's color names in appendix A. Please do not forget that CAPITALS are different letters than **small** letters, and spaces count.

Circles

Circles are easy to make in Python. Just type the word circle and put the size in parentheses after it. Here is an example:

```
circle(50)
```



This example draws a circle 100 pixels across. (Remember that pixels are the computer word for dots.) The number 50 is called the circle's **RADIUS**[3]. It is how far every part of the circle is from the center of the circle. Circles draw above the turtle's location in Python.

Try this clover for luck:

```
import turtle
win = turtle.Screen()
t=turtle.Turtle()

t.circle(150)
t.left(90)
t.circle(150)
t.left(90)
t.circle(150)
t.left(90)
t.circle(150)
t.left(90)
t.circle(150)
```

circle(radius) draws a circle
radius determines size

Olympic Challenge:

Try to draw the Olympic symbol

[3] The radius is equal to the distance from the center of a circle to its edge.

Angles

Using left(90) and right(90), it is easy to make rectangles and squares. The 90 in the parentheses is 90° pronounced ninety degrees[4]. The degrees entered in left and right tell the turtle how much to turn. A full turn so the turtle faces exactly the way it did before is a 360° turn. Try it using the code below.

```
right(360)
```

Half a 360° turn, or 180°, will make the turtle aim directly backwards. Likewise, half of that, 90° will make the turtle turn ¼ of the way around, or exactly what it would need to do to make a square. Try this program that makes an asterisk.

```
import turtle
win = turtle.Screen()
t=turtle.Turtle()

t.forward(50)
t.backward(100)
t.forward(50)
t.left(45)

t.forward(50)
t.backward(100)
t.forward(50)
t.left(45)

t.forward(50)
t.backward(100)
t.forward(50)
t.left(45)

t.forward(50)
t.backward(100)
t.forward(50)
t.left(45)
```

The left(45) command has the turtle turn 1/8 of the way around.

Triangle Challenge

See if you can make the 30°, 60°, 90° degree triangle show to the right.

[4] Our 60 minute hours, 60 second minutes, and 360 degree circles come from the Babylonians who used base 60 to do arithmetic. That meant they counted to 59 before putting a 1 in the "tens" position. See https://www.storyofmathematics.com/sumerian.html for more information.

More Turtle Commands

When people draw they often pick up their pens to draw in another location. The turtle can do that as well using **penup()** and **pendown()**. Try this example.

```
import turtle
win = turtle.Screen()
t=turtle.Turtle()

t.forward(10)
t.penup()
t.forward(10)
t.pendown()
t.forward(10)
```

penup()	turtle leaves no line
pendown()	turtle leaves a line

When people draw they often color their work in. The **begin_fill()** command will fill in any shape the turtle completes with the current color. To stop coloring things in use **end_fill()**. Be careful to use the underscore (shift + hyphen) when typing these. Here is an example.

```
import turtle
win = turtle.Screen()
t=turtle.Turtle()

t.color("red")
t.begin_fill()
t.forward(50)
t.right(120)
t.forward(50)
t.right(120)
t.forward(50)
t.right(120)
t.end_fill()
```

begin_fill()	color in
end_fill()	stop coloring in

Bullseye Challenge:

Make a bullseye with two rings around a filled center.

Comments

You can tell Python to ignore section of your code so you can remind yourself where things are or how they work. (These are called comments by programmers.) This can be very useful in a long and complicated program, or if someone else is likely to work with your code. There are two ways put in comments in Python.

The first way is to type **#** in front of the notes you want Python to ignore. The second is to type three quotes. Python will then ignore everything until the next set of three quotes. Here is that same program with notes.

```
"""
Red triangle program by Bright Brash and Costly Code (tm)
All rights reserved.
"""

import turtle
win = turtle.Screen()
t=turtle.Turtle()

#Change color here--------------
t.color("red")
t.begin_fill()
t.forward(50)
t.right(120) #60 degree internal angle made by 1/3 of 360
t.forward(50)
t.right(120)
t.forward(50)
t.right(120)
t.end_fill()
```

#	Ignore rest of line comment
"""	Ignore until next """ comment

Reading and Writing

Print

Most Python programs do not use the turtle screen, they operate apps and report on a text screen named the console using the print command. Print is easy to use. Simply type **print** and put what you want to print in the parentheses. Here is how that would work with numbers:

```
Print (500)
```

print ("text")	display on console
print(number)	

Because Python uses letters and words to store data, you need to put words in quotation marks, if you want them to print. Like this.

```
print ("This is the text screen.")
```

Print commands usually go to the next line automatically. Try this:

```
print ("I like eels,")
print ("Except as meals.")
```

Python 3 allows printing information across a line using commas. Here is an example.

```
Print(35, " Dogs ", 45, "  Cats ",1 " Pet Alligator")
```

Try this program that uses a command you have not had. (Be sure you indent the second line. Indentation connects commands in Python). Can you figure out what is happening?

```
for i in range(100):
    print(i)
```

One of the best things about computers is their ability to do simple things quickly and accurately. Here is the same program with a modification.

```
for i in range(100):
    print(i, end="")
```

The **end=** command tells Python what to do after it finishes printing the line. In this case, it prints nothing and stays on the same line.

20 Count Challenge:

Change the program above to count from zero to twenty.

Coloring Text in Python[5]

When you print, letters, numbers, and symbols are sent to the computer using a code called ASCII[6]. There are special commands that can be sent to change the way things print, as well, called escape codes. The escape codes are entered right into the print statement. Here is an example:

```
Print("\033[1;32;40m Bright Green   \n")
```

The above ANSI escape code will set the text color to bright green. The format is;

\033[Escape code, this is always the same (\033 is the ESC key).
1= **Style**, 0 for normal. Bold(1), underline(4) and blinking text(5) are all options.
32= **Text color**, 32, in the exampleis for bright green.
40m = Sets the **background color**, 40 is black. (47 is white.)

This table shows some of the available formats;

[5] With modification from http://ozzmaker.com/add-colour-to-text-in-python/
[6] ASCII stands American Standard Code International Institute.

Text color	Code	Text style	Code	Background color	Code
Black	30	No effect	0	Black	40
Red	31	Bold	1	Red	41
Green	32	Light Bold	2	Green	42
Yellow	33	Italic	3	Yellow	43
Blue	34	Underline	4	Blue	44
Purple	35	Blink	5	Purple	45
Cyan	36	Fast Blink	6	Cyan	46
White	37			White	47

```
print("\033[0;37;40m Normal text\n")
print("\033[2;37;40m Underlined text\033[0;37;40m \n")
print("\033[1;37;40m Bright Color\033[0;37;40m \n")
print("\033[3;37;40m Negative Color\033[0;37;40m \n")
print("\033[5;37;40m Negative Color\033[0;37;40m\n")
```

Here is a handy function. To print colored text, just **type cprint instead of print, a comma and a color number**. To use it, type cprint followed by an open parenthesis, the words you want to write in quotes, a color number, a style, and a closing parenthesis.You can leave color and style out, then cprint will print red text.[7]

Cprint does not go to the next line automatically, so if you want to go to the next line, you need to follow the cprint command with a regular print command.

To try it, copy and paste the function below to the top of one of your programs.

```
#Colored Text Function
#(Does not work in Thonny, in Trinket.io remove end='')

def cprint(txt,clr=4, style="0"):
# Style: 1=bold, 2=bold2,3=italic,4=underline,5=blink,6=fast blink
    if clr==0:  c="30" #0  = Black
    if clr==1:  c="34" #1  = Blue
    if clr==2:  c="32" #2  = Green
    if clr==3:  c="36" #3  = Cyan
    if clr==4:  c="31" #4  = Red
    if clr==5:  c="35" #5  = Magenta
    if clr==6:  c="33" #6  = Yellow
    if clr==7:  c="37" #7  = White
```

[7] The thing following the equal sign in the function is what is assumed, if you leave something out.

```
print('\x1b['+str(style)+';'+str(c)+'m'+txt+'\x1b[0m',end='')
#The line above prints in ascii ESC color sequence and text
#  ESC[ + color number + m    <the text to print> + ESC[0m
```

Here are some ways you could use it.

```
cprint("This is blue.  ",1)
cprint("This is green and bold.  ",2,1)
cprint("This is red.")
print()
print("That's better.")
```

Here is something to try out using that for loop command we used earlier:

```
#colors are numbers 0-7
for c in range(8):
cprint("testing",c)
    print()

#prints in red if nothing else is entered
cprint("xxxxx")
```

text answer=input("prompt")
number answer=int(input("prompt"))

Input

One of the great things about computers is their ability to take information, do computations and give out the result. Python has a great way to get the information it needs to help you, the input command. This is an example of how it works:

```
name1=input("Christian:  What is your name? ")
print("Christian:  Welcome aboard Captain ",name1,"." )
print(name1,":  Thank you Mr. Christian.")
print()
print(name1, ":  Raise the main and prepare to get under way,
swabbie.")
```

On the computer the screen prints out what is inside the parentheses of the input statement an then waits for you to enter information. After you finish, and press ENTER, it puts what you type into the variable on the left of the equal sign and then continues the program. The output of our program would look something like this:

```
Christian:  What is your name? Bligh<- The computer stops here until you finish typing
Christian:  Welcome aboard Captain Bligh.
Bligh:  Thank you Mr. Christian.
```

```
Bligh:  Raise the main and prepare to get under way, swabbie.
```

All the information you type is made into a string, THAT IS LETTERS, including the numbers you type, so:

```
n=input("Number?")
print(n, " squared equals ", n**2)
```

gives you an error.

When you want **to enter numbers, surround your input statement with an int()**[8] command. The **input()** command will bring the number you type into the program as letters and the int() changes those letters into the number you wanted to enter. Here is the earlier example fixed:

```
n=int(input("Number?"))
print(n, " squared equals ", n**2)
```

Here is another example; try this out. It uses a command you have not studied yet, IF. The double equal sign checks two things to see if they are equal. Everything that is indented after the IF only happens if a3 is 25.

```
print("Answer me these questions three err the other side you
see:")
a1 = input("What is your quest?")
print ("O, I have always wanted to ",a1,".")
a2 = input("What is your favorite color")
print("I think ",a2," is lovely too.")
a3 = int(input("What is the square root of 625?"))
if a3 == 25:
     print("You may pass.")
else:
     print("Sorry, you need to turn around now")
print("Next.")
```

Printing on the Turtle Screen

It is easy to add labels and instructions to your program in Python. Just move the turtle to the location you want to put your words and use the write command. Try this:

```
import turtle
```

write("Text")
displays text on turtle screen

[8] **Int** is short for integer. Math people define integers as the counting numbers, their negatives and zero. That means integers have no numbers after the decimal point. The computer can do integer math very quickly. If you need a decimal number, use float() instead of int().

```
win = turtle.Screen()
t=turtle.Turtle()

penup()
setposition(-40,50)
t.write("Hi friends.")
```

To add color simply put in a color command. You can also write out numbers and words stored in variables. Here is an example:

```
a="Programmers are really cool"
b=2+2
penup()
setposition(-40,50)
color("dark red")
t.write(a)
setpostition(-40,20)
t.write(b)
```

The full write command has many interesting features to explore. Here is how it works:

write(THINGYOUWRITE, ANYCOMBINATIONOFTHETHINGSBELOW**)**
move="true/false",
align="left/right/center",
font=("NameOfFont", PointSize, "bold/italic/normal"**)**

Here are some examples.
This will change the size and font. Be sure to Capitalize the font name:

```
write("This is 20 point", font=("Arial", 20, "bold"))
```

This adds on to what was written by moving to the end of the writing when it finishes:

```
write("First part", move="true")
write("SECOND PART")
```

Try this program:

```
#Turtle Write Program
from turtle import *
win=Screen()
```

```
t=Turtle()

t.write("testing 1,2,3 testing 1,2,3 testing 1,2,3 ", move="true")
right(90)
forward(120)
t.write("AFTER MOVE=TRUE")
t.setposition(40,-40)
t.write("20 point",font=("Arial",20,"normal"))
t.penup()
t.setposition(-100,50)
t.color("dark blue")
t.write("After color and penup")
```

Clock Face Challenge:

1. Make an analog clock face with turtle graphics.
2. Write a protractor program with labels.

Math

Basic Arithmetic

Science assumes the world is rational. People who believe this use mathematics to predict or simulate the real world. Python makes math easy. Find the >>> symbols in a box near your program. Type in this and press ENTER:

```
>>>2+3
```

Python does the math instantly. Try this using * for multiply, and see what happens:

```
>>>247*1.3
```

Here is a table of math operators.

Name	Symbol	Example	Result	In Math Written as	Order of Operation	Notes
Add	+	2+5	7	2+5	4	
Subtract	-	2-5	-3	2-5	4	(hyphen on your keyboard)
Muliply	*	2*5	10	2x5	3	X can be a letter so Python uses *
Divide	/	2/5	0.4	2÷5	3	
Exponent	**	2**5	32	2^5	2	2^5 = 2x2x2x2x2 = 32
Modulus	%	2%5	2	mod 5	3	Gives the remainder after division.
Parentheses	()	(2+5)	7	(2+5)	1	Forces contained to happen first

Try this one.

```
>>>4 + 1 / 5
```

Did you expect the answer to be 1? Python follows the same rules people learn in algebra. Math people call that rule the **order of operations**. Multiply and divide are done before add and subtract, so Python divided 1 by 5 getting 0.2 and only then added 4.

In math, if you want to change the order operations, and do add before multiply use parentheses. Any math in parentheses is done before everything else. Try this.

```
>>>(4 + 1) / 5
```

Exponents, also called powers are a way to write how many times some number is multiplied by itself. Here are some examples:

$3^2 = 9$ and is the same as 3x3 = 9 or 3**2 in Python
$5^4 = 625$ and is the same as 5x5x5x5 = 625 or 5** in Python
$7^3 = 343$ and is the same as 7x7x7 = 343 or 7**3 in Python

Use Python to find the following using **, the exponent or power operator:

1. 3^3
2. 9^2
3. 2^{10}
4. $4^{(2+1)}$
5. $2^{(5*2)}$

Here is a weird one for you. Try this one:

$16^{(1/2)}$

Did you get 4? That is because 16 to the power of one half is the same as asking what number multiplied by itself would be 16. That number is 4. Math people call that the square root. Part of the beauty of math is 343 ** (1/3) will give you the number that multiplied by itself three times gives you 343.

Area and Volume Challenge:

1. Write a program that calculates the area of a rectangle. (Remember that the area is equal to the length of the rectangle times the width.) Use an input command, so your program will calculate the area of any rectangle.
2. Ask for the radius of a ball and calculate its volume. (Remember the radius is the distance from the surface of the ball to its center.) Here is the formula for volume.

$$Volume = \frac{4}{3}\pi r^3$$

Arithmetic and Stars

Remember your turtle graphics? Mathematics can produce amazing art.

Polygons like triangles, squares, pentagon and stop sign octagons are just a group of lines made by the same forward command followed by a turn that is 360° (turning all the way around) divided by the number of sides in the shape. The turtle is just going in a circle with edges.

Here is a program that makes a triangle using the angle 360°/3 to make three 60° turns after.

```
#Triangle
import turtle
canvas=turtle.Screen()
t=turtle.Turtle()

t.forward (100)
t.left(360/3)

t.forward (100)
```

```
t.left(360/3)

t.forward (100)
t.left(360/3)
```

Try this pentagon program out. Pentagons have five lines and five turns.

```
#Pentagon
import turtle
canvas=turtle.Screen()
t=turtle.Turtle()

t.forward (80)
t.left(360/5)

t.forward (80)
t.left(360/5)

t.forward (80)
t.left(360/5)

t.forward (80)
t.left(360/5)

t.forward (80)
t.left(360/5)
```

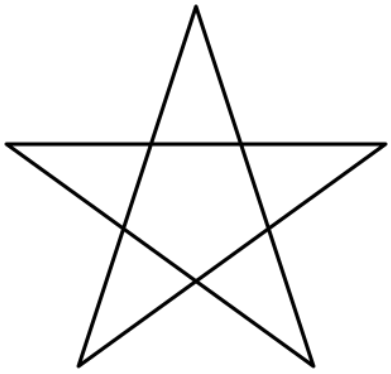
Stars are like circles too, but the turtle will turn and make a line that connects to a point on the other side. Remember that a 360° turn turns the turtle all the way around so it is aimed exactly the same direction it was before. That means 180°turn will turn the turtle so it is facing backwards. To make a star, The turtle must turn completely around minus the amount it needs to make the angle divided by two because you need revolve two times. So the angle to turn is

$$turn = 180° - \frac{360}{points \times 2}$$

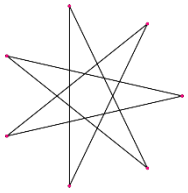
Here is a program that makes a star with five points.

```
#Star 5 Point
import turtle
canvas=turtle.Screen()
t=turtle.Turtle()

t.forward (80)
t.left(180-360/(5*2))
```

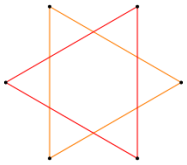



```
t.forward (80)
t.left(180-360/(5*2))
t.forward (80)
t.left(180-360/(5*2))
t.forward (80)
t.left(180-360/(5*2))
t.forward (80)
t.left(180-360/(5*2))
```



Seven Point Star Challenge:

Make a 7-pointed star.
Use an input command to make any star with an odd number of points.



Six PointStar Challenge+:

Even pointed stars are actually two stars offset. For big credit, make a 6-pointed star.

Random Numbers

Oddly enough, there is no such thing as a random number for a computer. Some people take this idea further, believing the whole world is like a computer, and all events are planned out.[9] Computer that start up with certain data in them always end up with the same numbers when they try to make a random number. Computer scientists call random numbers **pseudo-random numbers** for that reason. Python loads the exact time in milliseconds that you start the program into a number scrambler to get a series of numbers that behave like random numbers.[10] To use it, import the random function from the random library. Here is an example:

```
from random import random

print (random())
print(random())
print(random())
print(random())
```

random() is a random number

[9] Called fatalists, these people believe there is an eternal plan or destiny that cannot be changed. The things people decide to do were just part of the plan to begin with. Some believe the plan was made by a benevolent all knowing all powerful God, others that fate is arbitrary, often cruel. One of the two is definitely the case with computers. They just run pre-planned programs continually and never act apart from their programming.
[10] Strangely, you, the human, end up being the random event that all random numbers can be built upon by running the program a random number of milliseconds after midnight.

The random() function produces a number between zero and one. It is never zero and it is never one. By multiplying and adding, random() will make any kind of random number you want. Here is a program that rolls a six-sided die.

```
from random import random

#die roll
r=int(random()*6+1)
print (r)
```

Try this program:

```
#Addition Quiz
from random import random

n1=int(random()*9+1)
n2=int(random()*9+1)
print("What is ",n1,"+",n2, end="")
a=int(input("?"))

if n1+n2 == a:
    print("Right.")
else:
    print("Wrong.")
```

Random Challenge:

1. Write a program that prints a random number 1-100.
2. Write a program that adds 3 dice and prints the sum.

Loops

The FOR Loop

Here is a really neat way to do lots of work using Python. Try this program out:

```
for i in range(100):
     print("This is fun.")
```

The number in the range() command is how many times it runs the indented text after the colon. Start up turtle graphics and try this program:

```
import turtle
win = turtle.Screen()
t=turtle.Turtle()
```

for [counter] **in range**([count to this])**:**

```
for i in range(4):
    forward(100)
    right(90)
```

Now try this:

```
from turtle import *
wn=Screen()
t=Turtle()

t.speed(0) ← Speed(0) makes the turtle go top speed where 1 is slow and 9 is fast.

for i in range(18):

    t.color("dark red")
    t.forward(150)
    t.left(170)

    t.color("dark blue")
    t.forward(150)
    t.left(170)
```

Circle Pattern Challenge:

Use a for loop to draw a pattern using different colored circles.

Counting using for loops

Remember the example:

```
for i in range(10):
    print("This is fun.")
```

The letter **i** in the for loop is a **variable**[11] that the for loop uses to count. Programmers call these counting variables**indexes**. You can use the index in your program. Try this program that counts from 0 to 99.

```
for i in range(100):
print(i)
```

[11] Math people uses variables in the place of numbers with an unknown value. In programs, variables are the name of a location in memory that stores data. They are like named boxes that you can put things into.

Python counts a bit differently than people do, **IT STARTS COUNTING WITH THE NUMBER ZERO** and ends one number before reaching the number in parentheses. You can fix this by changing the program as shown here:

```
for i in range(100):
    print(i+1)
```

Starting and Stopping Loops

The word range is also a special command called a function[12]. A function is a small program within a program. Here are some of the things it can do.

You can tell the for loop what number to start counting by. In this example, the for loop counts from 200 to 300.

```
for i in Range(200,301):
    print(i)
```

for[c] **in range**([s],[e],[i]):
- c – a variable to count with
- s – The number to start with
- e – End before this number
- i – The number to count by

(Remember that Python stops before reaching the last number. That is why you need to put 301 rather than 300 to count up to 300.) You can also count by numbers other than one and even count backwards. Try this program.

```
for i in range(5, 101,5):   #Counts by fives from 5 to 100
    print(i)
```

Now try this one.

```
for i in range(100,0,-1):   #Counts backwards from 100 to 1
    print(i)
```

Milk Song Challenge:

Write a program that prints out the song to the right using for loops and print commands. Be careful

Milk Song

100 bottles of milk on the wall,
100 bottles of milk,
if one of those bottles should happen to fall,
there'd be 99 bottles of milk on the wall.

99 bottles of milk on the wall,
99 bottles of milk,
if one of those bottles shou …

2 bottles of milk on the wall,
2bottles of milk,
if one of those bottles should happen to fall,
there'd be 1 bottle of milk on the wall.

1 bottle of milk on the wall,
1 bottle of milk,
if that bottle should happen to fall,
there'd be no more bottles of milk on the wall.

[12] Functions are used in math, as well. In math, they are followed by p
like they are in Python. They represent complex formulas that produc
the vertical line test.) By just writing the name of the function and put
becomes equal to the answer you would get by doing the complex ma
functions are modeled after math function, except sometimes they pr
Python and simplify programs.

with the ending of the song. There are changes in the words.

5 Decisions Using Conditionals

IF and Progarm Control

Remember the program you saw earlier that had the bridge keeper decide if you could pass?

```
#Troll Bridge Program

print("Answer me these questions three ere the other side you see:")
a1 = input("What is your quest?")
print ("O, I have always wanted to ",a1,".")
a2 = input("What is your favorite color")
print("I think ",a2," is lovely too.")
a3 = int(input("What is the square root of 625?"))
if a3 == 25:
print("You may pass.")
else:
print("Sorry, you need to turn around now")
print("Next.")
```

In it the computer found out if the traveler knew the square root of 625. Everything indented after the **:** happens only if the answer they type is 25. In this case there is only one line, so the computer tells the person he or she can pass.

Everything indented after the **else:** happens only if the number entered is not 25. Once again, it is only one line. He or she is told to turn back. WHEN THE INDENTATION STOPS, THE PROGRAM IS DONE WITH THE IF COMMAND. When the indentation stops, the code is always executed, just like it was before the if statement.

If statements can check for more than just equality. Look at this chart.

	if operator	Example
Equal	==	if a == b:
Not Equal	!=	if a != b:
Greater than	>	if a > b:
Less than	<	if a < b:
Greater than or Equal	>=	if a >= b:
Less than or Equal	<=	if a <= b:

Here is a program that will check numbers to see if they are positive, negative or zero.

```
#Positive Negative Program

n=int(input("->"))
if n>0:print("positive")
if n<0:print("negative")
if n==0:print("zero")
```

Python lets you put the command that will run if the if statement is true right after the colon, when there is only one of them.

Here is another program that checks to see if the numbers are good before dividing. Remember, in math, you cannot divide by zero. If you do, you get an infinitely large number, and infinity is an idea, not a number. The second interesting thing about the program is that it uses **float()** instead of **int()**. That way it can change the strings of letter coming in to decimal numbers.

float(string) convert to decimal num

```
#Divide Program

n=0.0
d=0.0

print("This program divides two numbers.")

n=float(input("->")) #Numerator (number to divide)
d=float(input("->")) #Denominator (Number to divide by)
if d==0:
    print("You cannot divide by zero.")
else:
    print ("= ", n/d)
```

Here is a game called the guessing game. It uses the random number generator. Try it out.

```
"""
Guessing Game program
"""
from random import random

n=int(random()*100+1)  #This is the number to guess

print("I have a number between 1 and 100")
print("See if you can guess it in two tries.")
print()
guess1=int(input("What is your first guess?"))

if guess1==n:#-----Try 1------
    print("Got it.")
    print("You win.")
else:
    difference=n-guess1

    ab=difference #take the absolute value for the hint
    if ab<0 : ab=-ab

    print ("You are off by ",ab,".")
    guess2=int(input("What is your second guess?"))

    if guess2==n:#-----Try 2------
        print("You win.")
    else:
        print("You lost.  The number was ",n,".")
```

Here is another example. This program finds all the factors of a number typed in by dividing and comparing to the division without decimals using int(). People who go into math and science are always wondering if numbers are prime so their theories can be proven.

```
# Factoring program

n=int(input("Enter a number to factor."))

half=int(n/2)
# the most a factor could be is half the number

print()
print("Factors of ",n,".")
for i in range(half):
    f=i+1# for goes from 0 to half-1, this corrects
    if (n/f)==int(n/f):
        print(f,end=" ")
```

Prime Number Challenge:

Change the factoring program so it does not list out 1. Mathematicians never list 1 as a factor
Change the factoring program so it prints PRIME if there are no factors.

While Loops

For loops can do magical things, but they need to have a stopping number, and sometimes you do not know when to stop ahead of time. When that is true, it may be time to use Python's while command. A while command loops as long as a condition is true. Conditions work the same way they do in if statements.

```
# Summation Program

t=0.0    #Floating point total for decimal numbers
n=0.0    #Number to add to the total
alpha="1" #Alphabetic answer, used for the loop
#Set to 1 so the loop runs the first time.

print("Summing program")
print("Type numbers to be summed")
print("Press enter twice to get the sum.")

while alpha!="":#alpha nothing means user just pressedENTER
    alpha=input("->")
    if alpha !="":
        n=int(alpha)
    else:
        n=0
    t=t+n

print("Total = ",t)
```

Average Challenge:

Math people tell us the average is the total divided by the number of things add to make the total. For example, 3+5+7+9 totals to 24, and because four numbers were added, the average is 6. Modify the summation program above to average the numbers. **MAKE SURE YOUR PROGRAM DOES NOT DIVIDE BY ZERO.**

Here is another example of a while statement; this one uses turtle graphics, and exits using **break**. Break will exit a for loop or while loop and continue your program with the commands that come after the loop. Another handy command, **continue**, will skip the rest of the loop.


```
#  Slinky Program

from turtle import *
canvas=Screen()
t=Turtle()
t.speed(0)

#Get into position ------
t.penup()
t.backward (200)
t.left(90)
t.backward(100)
t.right(90)
t.pendown()

x=0  #horizontal position
dx=5 #rate of travel
while 1:#1 is always true
    t.circle(100)
    t.penup()
    t.forward(dx)
    x=x+dx
    t.pendown()
    if x>400:break
```

break	exits a loop
continue	skips the rest of the loop

6 Functions

Defining a function

1. Start with the **def** command def stands for define function.[13] Def tells the computer you are to give it the rules for what it will do when someone runs the function.[14] When a function is run, programmers say it was **called**. A space always follows the def command.
2. The **name of the function** comes next names follow the same rules that variable names follow it is good technique to write names that are as descriptive as possible.
3. Next there is a set of **parentheses** sometimes there will be variables listed between the parentheses that give the function the information it needs.[15] That will be discussed later.

[13] Math people will tell you a function is like a black box you send things in and only one answer comes out.
[14] Programmers have a big word for commands that care that create sub-programs a call them **constructors**.
[15] The variables between the parentheses in the def statement are substitute names the function uses for copies of data from the main program. These variables are called **parameters**. The function call is called an **instantiation**. Later when variables are passed between the parentheses in the function call, they are given a different name. They are called **arguments**.

4. The def line ends with a **colon** after the colon the parentheses. Just like *if*, *for*, and *while* all the commands in the function will follow the def line and are indented. Python tells what is included in the function by its indentation. It is vital that you indent accurately.

Here is the form:

```
def FunctionName ():
     Command
     Command
     …
```

Using a function

To use a function just type its name followed by parentheses but no colon.

Here is the form:

```
FunctionName()
```

You have used functions before. In fact, some of the first programming you did used functions that are built into Python. Forward(), int(), input(), and print() are all functions. Being able to create functions allows you to add capabilities to Python.

Here is an example of a function that adds A and B and prints the sum.

```
def addAnB():
     c = a + b
     Print(c)
a = 7
b = 8
addAnB()
```

here is a function that writes the numbers 1 to 100.

```
def count100():
     for i in range(1,101):
          print (i)
```

Functions are very useful for simplifying problems. Code that is used many times can be written as a function. Doing this saves programming time and makes the code less prone to error.

One of the biggest advantages of functions is that programmers can change code that is used throughout their program by changing it in one location.

Another nice feature of functions is the **return** command. When you put the return command in a function, it does two things: It makes its name equal to whatever follows the return in the statement, and it immediately ends the function and goes back to the main program.

Here is an example:

```
Def circumference ():
     c=2*3.14159*r
     return c← circumference now equals c

r=12
print ("A circle with a radius of 12 feet")
print("would be ",circumference(), " feet around.")
```

Challenge: Area Function

Using the example above, write a program with a function named ***area*** that asks for a circle's radius and prints out the area of the circle.

Challenge: Factorial Function

The factorial of a number is equal to all the numbers from 1 to the number multiplied. Factorials are written with an explanation point, so 4!=1x2x3x4, or 24.[16] Write a function that calculates the factorial of any number stored in a variable named n.

7Animation

Formula for Simple Animation

1. Show it.
2. Wait > 1/30 sec.
3. Erase it.
4. Change it.
5. Do it again.

Using Zelle graphics in Python

Years ago, film makers showed a series of pictures quickly to make movies. They discovered that the human eye cannot see things that happen faster than 1/30 of a second, so when a picture is quickly switched for another picture that is slightly different, viewers see it as motion. Make quick changes to graphics, we are going to have to leave Turtle Graphics behind, and take a different approach.

[16] Factorials are used for many things in statistics including combinations and permutations.

A professor of mathematics named Dr. John Zelle wrote a graphics package for Python that you can download and put with your Python programs. It gives you commands that make lines, ellipses and many other handy items. Here are some things you need to do to get ready:

1. Search up **Zelle's graphics for Python**. Download **graphics.py** and save it to the place you save your Python programs.[17]
2. Include these two lines at the top of your program before you use his graphics.

```
from graphics import *
win = GraphWin("Output", 640, 480,1)
```

The second line sets up a graphics window that is 640 x 480. Modify these number, if you like, to make the screen whatever size you want. You can have more screens by just having more commands like the second one in your program.

Zelle's graphics module works by making drawings into objects. Objects have properties, like color, height and visibility. Objects also have methods. Methods are little programs built into the object itself. They run effecting object itself, and making it do things.[18]

- The **move** method will move the object to whatever (x,y)[19] you want.
- The **draw** method will put the object on the screen. (NOTHING SHOWS UNTIL YOU DO THIS.)[20]
- The **setFill** method colors objects in.
- The **setOutline** method outlines the object in any color you want.

Quick Summry of commands from Zelle's graphics.py	
aPoint = **Point**(x,y)	Color a pixel (dot)
aLine = **Line**(Point(x1,y1), Point(x2,y2))	Draw a line from the two points
aCircle = **Circle**(Point(x,y), r)	
aRectangle = **Rectangle**(Point(x1,y1), Point(x2,y2))	Upper left corner to lower right corner
anOval = **Oval**(Point(x1,y1), Point(x2,y2))	Oval contained by the rectangle
aPolygon = **Polygon**(Point(x1,y1), Point(x2,y2), Point(x3,y3))	Fill in the object drawn by the points
message = **Text**(Point(x,y), "Hello!")	Center point and text

[17] Dr. Zelle of Wartberg College made a module of graphics commands for students that can be installed by just having the file saved in the same place students store their programs. He and Wartberg College have graciously made this module available, free, to anyone who wants it. See https://knowledge.kitchen/John_Zelle_Python_Graphics_Module for information about Dr. Zelle and his graphics module.

[18] See https://bouwsma.neocities.org/docs/graphics.pdf for a full list of properties and methods for objects.

[19] In math, people use **x** to represent the horizontal position of an object, and **y** to represent the vertical position. Computer monitors are strange. **They reverse y**, the vertical position. They place the object lower as y gets larger.

[20] Waiting until all methods and properties are in makes the computer execute faster. This is part of the basic design of Python. If you want to simplify it, simply put your commands in a function.

inputBox = **Entry**(Point(x,y), size)	inputBox.getText() contains the text
flowerImage = **Image**(Point(x,y), "flower.gif")	Center and file
aCircle.setFill(**color_rgb**(r, g, b))	Red, Green, Blue (0-255)
IN THE TABLE ABOVE, **X** REPERESENTS THE DISTANCE FROM THE LEFT SIDE OF THE WINDOW **Y** REPRESENTS THE DISTANCE FROM THE TOP OF THE WINDOW **R** IS THE RADIUS OR THE SIZE **R,G,B** IS THE AMOUNT OF RED, GREEN, AND BLUE 255 IS BRIGHTEST, 0 IS COLOR NOT PRESENT SEE WARTBURG COLLEGE FOR MORE COMPLETE INFORMATION	

When you use graphics in Python, you put circles, lines and other shapes in variables and use each variable's draw method to show them in a window, the way the turtle used a window. Try this program out.

```
from graphics import *
win=GraphWin("Window Name Here", 400,400)

#draw a point-------------------------
pt=Point(100,50)
pt.draw(win)

#draw a circle----------------------
cir=Circle(pt,50)
cir.draw(win)

cir2=Circle(Point(200,200),25)
cir2.setOutline('red')
cir2.setFill('blue')
cir2.draw(win)


#draw a line------------------------
ln=Line(Point(100,120),Point(20,150))
ln.draw(win)
```

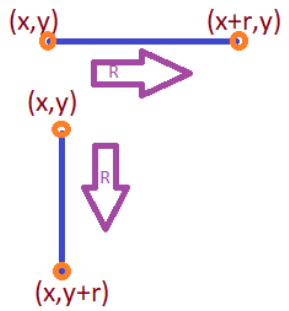
Truck Challenge:

Use ideas from the program above to draw a truck.

Try this program. It uses a for loop with Zelle graphics to make inset circles by increasing the radius.

```
from graphics import *
wind=GraphWin("Window Name Here",400,400)
```

```
for r in range (5,200+5,5):
    c=Circle(Point(200,200),r)
    c.draw(wind)
```



Now try this program that draws lines. It adds and subtracts r as it changes from both the horizontal and vertical end points of the lines.

```
from graphics import *
wn=GraphWin("Cheverons",400,400)

for r in range (2,200+2,2):
    line1=Line(Point(200-r,200-r),Point(200-r,200+r))
    line2=Line(Point(200-r,200-r),Point(200+r,200-r))
    line1.draw(wn)
    line2.draw(wn)
```

100 Squares Challenge:

Use your knowledge to draw 100 inset squares

Cyber Eye Program

The combination of graphics and looping can produce some amazing pictures. Try this program out and see what you think.

```
from graphics import *
win=GraphWin("Cyber Eye",600,600)

for m in range(5,600+5,5):
    L1=Line(Point(600-m,1),Point(600,600-m))
    L2=Line(Point(1,1+m),Point(1+m,600))
    L1.draw(win)
    L2.draw(win)
```

One line slides across the right and down, the other slides up as it moves left.

Text on a Graphic Window

Zelle graphics has a Text command that can make objects out of text and display them if you give it a center point called an anchor and the text to print. Try this example.

```
from graphics import *
sheet = GraphWin("Text Example",400,400)

textObject = Text(Point(250,100),"This text is centered at 250,100")
```

```
textObject.draw(sheet)
```

The text command has options that allow you to do just about anything you want. Try this program out that changes the font, color, size, and style of text.

```
from graphics import *
page = GraphWin("Text Example",400,400)

t1 = Text(Point(150,50),"normal")
t1.draw(page)

t2 = Text(Point(150,100),"blue")
t2.setTextColor("blue")
t2.draw(page)

t3 = Text(Point(150,150),"20 Point")
t3.setSize(20)
t3.draw(page)

t4 = Text(Point(150,200),"Font  changed to Courier")
t4.setFace("courier")
t4.draw(page)

t5 = Text(Point(150,250),"Bold and italic")
t5.setStyle("bold italic")
t5.draw(page)
```

Some Zelle Graphics Text commands			
	Action	**Command Example**	**Options**
1	Color of Text	`ob.setTextColor("red")`	red, green … See all optionshere.
2	Point Size	`ob.setSize(18)`	5 to 36 point
3	Change Font	`ob.setFace{"arial")`	helvetica,courier,times roman, arial
4	Bold / Italic	`ob.setStyle("bold")`	Bold, italic, normal
5	Text to Display	`ob.setText("anything")`	

Moving Circle Program

Here is a program that moves a circle across the screen using Zelle's **move** method. If animation is simple enough, Dr. Zelle does the work for you when you use **move**.

```
#Simple Animation
# Uses Zelle Graphics

from graphics import *
win = GraphWin("Moving Circle", 640, 480,1)
```


```
x=100[21]
y=100
dx=1[22]
delay=100000

c = Circle(Point(x,y), 10)#Make an object and draw it
c.draw(win)

while 1:
x=x+dx#  Plot new position
c.move(dx,0)#move the object in c

    for d in range(delay):pass
```

X is how far from the left
Yis how far down

DXis its speed left and right
DYis its speed up and down

pass is a command that does nothing but use up time

Try changing the number delay is set to at the beginning of the program. How fast can it go before no one can see it moving?

Now try changing **dx**, the number of pixels circle **c** moves each time through the loop. How many pixels can you skip before the circle no longer seems to move smoothly?

Here are some commands to remember when using Dr. Zelle's graphics package.

	Parts	Example
Point	Object = Point(x,y)	p = Point(x,y) p.draw(win)
Circle	Object = Circle(Point(x,y), radius)	c = Circle(Point(100,150),50) c.draw(window)
Line	Object = Line(Point(x1, y1), Point(x2, y2))	L =Line(Point(x,y),Point(x+240,y+100)) L.draw(win)
Rectangle	Object = Rectangle(Point(x1, y1), Point(x2, y2))	L = Rectangle(Point(x, y), Point(x+240,y+100)) L.draw(win)

Edge of Window Programming

Animation is amazing, but the object looked quite basic, and it went off the screen. The setOutline() and setFill() methods can make the circle a proper object that is easy to see and

[21] Math experts like to use x and y for horizontal and vertical. In programs, you could name them whatever you want. Please remember y goes up in math, but computer monitors make y go down.
[22] Just like x and y are special to math professors, dx and dy have a special meaning. Dx is horizontal speed, or the rate of change for x; dy is the vertical speed or the rate of change up and down.

looks more like a real object. **Methods**, once again, are commands that are part of the object itself. Remember, **TO USE A METHOD, WRITE THE OBJECT NAME, A DOT AND THE METHOD AFTER IT**, like this:

earth.setFill("blue")

When an object touches the edge of the window one of four things can happen:

.

- It can **bounce**, by reversing its motion,
- **wrap** and appear on the other side of the screen,
- **interact** with the edge by crashing or sticking,
- or it can invisibly **travel** off the screen.

In the following example, we will do the following:

1. We will give the circle an outline and color.
2. We will give it a **dy**, that is a vertical motion.
3. When it reaches an edge, we will reverse its motion, so it bounces.

```
# Moving Circle with Bounce on Edges
# Uses Zelle Graphics

from graphics import *
win = GraphWin("Output", 640, 480,1)
#                          Horz,Vert

x=100#Position
y=100
dx=3#Speed
dy=1
delay=100000#Wait for humans to see

c = Circle(Point(x,y), 10)#Make an object and draw it
c.setOutline('black')
c.setFill('red')
c.draw(win)

while 1:
    x=x+dx#  Plot new position
    y=y+dy
if x>640 or x<1:#Horizontal bounce -> reverse the motion
dx=-dx

if y>480 or y<1:#Vertical bounce
dy=-dy
```

Ob.setFill() fills in Zelle objects
Ob.setOutline() gives them a border

```
c.move(dx,dy)#move the object in c

for d in range(delay):
    pass
```

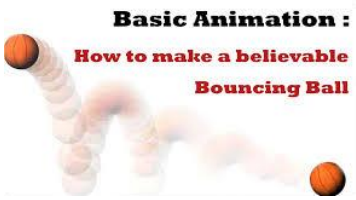
Two Circle Challenge:

Add a second circle that moves independently. **(HINT, IT WILL NEED ITS OWN X, Y, DX, AND DY.)**

8 Vector Physics

Adding Gravity

Our friends in the physics lab can tell you that gravity is an acceleration of 9.8 meters per second added to the downward speed of an object every second. To add gravity, simply put a variable for it at the top of the program where you set other variables and add the gravity to the object's vertical motion. Notice, you add, not subtract. Higher numbers are lower on the screen in Zelle graphics. Here is the same program with gravity added.




```
# Bounce Program With Gravity
# Uses Zelle Graphics

from graphics import *
win = GraphWin("Output", 640, 480,1)
#                        Horz,Vert

x=100#Position
y=100
dx=3#Speed
dy=1
delay=60000#Wait for humans to see
g=.003#Set gravity to what looks right.

c = Circle(Point(x,y), 10)#Make an object and draw it
c.setOutline('black')
c.setFill('red')
c.draw(win)

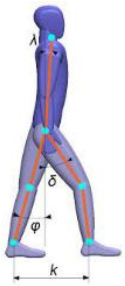
while 1:
dy=dy+g# <-- gravity adds to downward motion
x=x+dx#  Plot new position
    y=y+dy
    if x>640 or x<1:#Horizontal bounce -> reverse the motion
       dx=-dx
```

```
if y>480 or y<1:#Vertical bounce
   dy=-dy

c.move(dx,dy)#move the object in c

for d in range(delay):
    pass
```

Biomechanical Movement



There are two main ways to make people or creatures walk, crawl or slither. One way is to create many pictures of the creature and show them one after another. This method of drawing is called raster graphics. We will take up that topic in another section. A second way to simulate movement is through vector mathematics. We will program the simplest case possible, a stick man.

Because you eventually want your stick man to move be sure that each element has something you can change added.

Instead of :

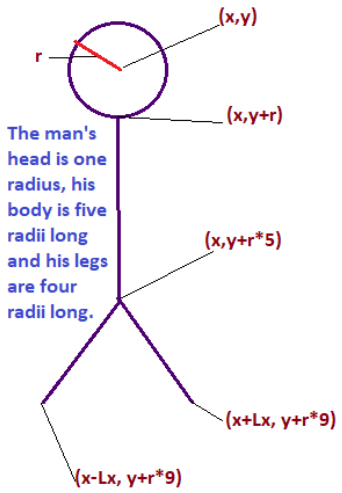
```
C= Circle(Point(100,120),20)
```

Write:

```
C=Circle(Point(100+x,120+y),20)
```

In the first example, the circle will always be drawn centered at (100,120) when it is in a loop. In the second example, the circle will change location **if x and y change**.

Step 1: Drawing a Stick Man



Our stick man will have a circle for a head and lines for a body and legs. Add arms, if you want to.

When moving a complex object, like our stick man, it is a good idea to locate all the parts relative to a key part. In this case, the man will move with his head. That means his body and legs will need to be

mathematically drawn so they line up with wherever the head is. Let's start with the following:

1. His **head** is at (x,y), and has a radius r. His position will change as x and y are changed, just as are circle did earlier.
2. His **body**and legs will move with (x,y) and be sized using r.[23] Just as the circles did in the for loop.
3. His **legs** will move at the same rate he moves forward (relative to his body.)

Stick Man Challenge:

Using the drawing above, make an app that draws a movable stick man.

Functions Parameters and Scope

Python functions allow you to make new commands and features for the Python language itself. Functions are like whole self-contained programs ontothemselves.

Remember that functions are declared using the def statement. Just like loops and conditionals, the whole function needs to be indented after the def statement. Click here for a review.)

The first thing you need to know before you can use functions well is how to get information from the main program into your function. Here are some rules and an example:

- **Variables from the main program:**Python assumes that variables used outside the function are the same variables with the same values inside the function. If an external variable is changed, the change in the function, the changes happen throughout the program.
- **Local variables:** If a variable is created inside the function, the main program does not know anything about it – even if it has the same name as the outside variable.
- **Parameters:**Parameters are typed in the parentheses that follow the name of the function in the def statement.They exist only in the function. Python copies numbers and variables from the place you call the function into these names. When the function ends, the parameters are discarded, and computer memory is freed up for other uses.When the function is called, the parameters are called **arguments**.They are real variables and numbers that are copied into the parameters. They help to customize functions. You already used them when you used the int() function.

```
def test(x,y):          ← Parameters x,y in function definition
    a=5                 ← Local variable a
```

[23] Using r to size all the parts of the stick man will allow the programmer to make him smaller as he goes off into the distance and larger as he comes closer for 3-D effects.

```
    print("x=",x)
    print("y*a=",y*a)

a=10                    ← Global variables a,b in main program
b=20
test(b,30)              ← Function call b and 30 are copied into x,y
print(a,b)
```

Output:

x= 20

y*a= 150

10 20

Sometimes a function needs to create a global variable for the main program because it may not exist already. In that situation, use a global statement in your function. Here is an example:

Global variables are variables used everywhere in the program. When you change a global variable in a function it changes for the outside program as well

```
def enemyCircle():
    global x,y,r
```

Here is an example of parameters in the function definition.

```
def enemyCircle(x,y,r):
```

When **parameters** are used in the parentheses, the variables, like (x,y,r) in the previous example, the variables x,y,r**exist only in the function**. Python copies numbers and variables from the place you call the function into these names.When the function ends, the temporary variables are discarded.

Programmers call the places variables exist the variable's**Scope**. **Global variables exist everywhere.** Variables declared inside a function or the parameters between a function's parentheses are called **Local**. Creating a function inside another function is good programming practice butwill create complex variable scopes where some variables do not exist in the parent function.

Programmers like to use functions and local variables. Using functions and local variables allows them to work in teams and not worry about what their coworkers named variables or how the function works internally.When teams assemble a program out of functions, they try to write each function so no changes are made in the way the main program operates. A program that changes the color of an object and does not change it back would be called **rude** by programmers.

In this function, a circle is drawn at (200,300) that has a radius of 10.

```
horizontal = 200
vertical = 300
rad = 10
enemyCircle (horizontal, vertical, rad)

def enemyCircle(x,y,r):
     circle(Point(x,y),r)
```

Stick Man Function Challenge:

Make your stick man into a function

Step 2: Making the stick man move

Making your stick man move is the same as making a circle move, as long as both are tied to the same x and y location. Check out this example. It moves a circle and a line

```
from graphics import *
wn=GraphWin("Two objects",400,400)
x=1
y=200
r=20

for x in range (400):← Move to a new location
    line1=Line(Point(x,y-r),Point(x,y-50))← Draw objects
    cir=Circle(Point(x,y),r)
    cir.setFill("blue")
    line1.draw(wn)
    cir.draw(wn)

    for d in range(200000):pass← Wait so we can see it

    line1.undraw()                    ← Erase objects
    cir.undraw()
```

Here is a working program that draws a stick man.

```
from graphics import *      # S T I C K   M A N
win=GraphWin("BioMechanical",800,350)

x=30            #Initial position across
y=200           #Initial position down
r=5             #Size
```

```
dx=1            #Sideways speed
dy=0            #Up and down speed
Lx=3            #Leg apartness
dLx=dx          #Leg speed
LxMax=6         #Maximum leg stride
delay=700000    #How long for people to see


for i in range(750):
    head=Circle(Point(x,y),r) #Draw the man
    body=Line(Point(x,y+r),Point(x,y+r*5))
    Leg1=Line(Point(x,y+r*5),Point(x+Lx,y+r*9))
    Leg2=Line(Point(x,y+r*5),Point(x-Lx,y+r*9))
    head.draw(win)
    body.draw(win)
    Leg1.draw(win)
    Leg2.draw(win)

    for d in range(delay):pass #Wait for humans to see

    head.undraw() #Erase the man
    body.undraw()
    Leg1.undraw()
    Leg2.undraw()

    x=x+dx          #Move the man
    Lx=Lx+dLx
    if Lx>LxMax or Lx<-LxMax: dLx=-dLx #Reverse legs
```

Stick Man Movement Challenge:

Using the drawing above, and the example program, make an app that draws a movable stick man. Adjust numbers so your program works well on your computer.

Moving Objects in Real-Time[24]

Earlier, we learned to give the computer information using the input command. Remember that the user would type in information while the program waited. When the user pressed Enter, the program continued. In a real-time game, the enemy should still be shooting while you decide what to do. In a real-time program, the computer is still monitoring and updating stock prices while you check your email.

[24] Real-time is a term programmers use to describe programs that continue to operate while waiting for the user to enter information.

Dr. Zelle provided a method named checkKey() that check the keyboard buffer[25] without stopping the program. To use checkKey(), type a variable with an equal sign, then type the name of the screen and a period followed by checkKey(). Here is an example:

```
Keypress = win.checkKey()
```

When this statement is executed, win.checkKey() becomes equal to the key the user just pressed, and that key value is put into the variable named Keypress. CheckKey() then erases the information stored in the keyboard buffer so it is ready for a new key press. If the user has not pressed anything checkKey is equal to "" (nothing).

Using checkKey()

Here is an example for you to try:

```
#file checkKey
from graphics import *
win=GraphWin("xx",400,400)

k=""

while k!="Escape":# Stop the program with ESC

k=win.checkKey()
    if k == "": print(".", end="")
    else: print("-->","k","<--")

win.close()
```

This program prints dots while it waits for you to press a key. When you press a key, it prints it.

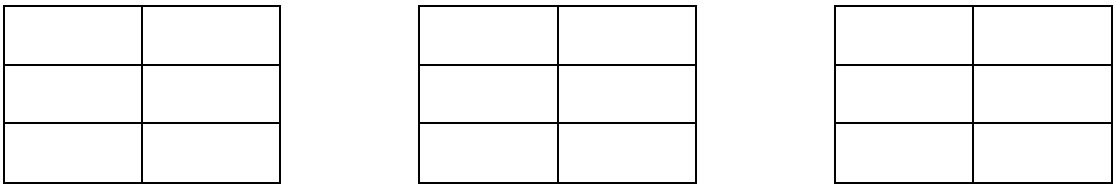
Zelle's graphics package prints out words for special keys including Escape. Try out things like shift, function keys, the space bar and others. Fill out the table that follows with your results:

Key	Result

Key	Result

Key	Result

[25] A buffer is a temporary storage location in fast memory. In this case, it is the place the operating system puts the key that was pressed, while the computer looks up what to do with it.



Moving Objects by Keypress

Earlier, we made a circle move and even bounce against the edges of the window. In this next program, we will control the movement of a circle, an instead of bouncing it, we will make it wrap.[26]

Here is an earlier example where we made a circle bounce. Type this in, or load your saved program. The code that makes the circle bounce is highlighted, and will need to be removed for the circle to wrap.

```
from graphics import *#Bounce Program
win = GraphWin("Output", 640, 480,1)

x=100 #Position
y=100
dx=3#Speed
dy=1
delay=100000#Wait for humans to see

c = Circle(Point(x,y), 10)#Draw it
c.setOutline('black')
c.setFill('red')
c.draw(win)

while 1:
    x=x+dx#  Plot new position
    y=y+dy
if x>640 or x<1:#Horizontal bounce -> reverse the motion
dx=-dx

if y>480 or y<1:#Vertical bounce
dy=-dy

    c.move(dx,dy)#move the object in c

    for d in range(delay):
        pass
```

[26] If you are wondering what wrapping is, follow this link.

Change1: To wrap the circle, we will need to check four conditions: the too high, too low, too far to the right, and too far to the left. When the circle goes too far add or subtract the size of the window to bring the circle to the other side. Here is code that will do that for a window that is 640 across by 480 down.

```
if x>640 :x=x-640 #Wrap it to the left
if x<1:   x=x+640 #Wrap it to the right
if y>480: y=y-480 #Wrap it to the top
if y<1:   y=y+480 #Wrap it to the bottom
```

Change2: Change dx and dy, the rate of speed across and down, to zero, so the circle is not moving when the program starts.

Change3: Because the circle's movement is complex, it is easier to draw and undraw it, rather than move it using the **move** method. Use the undraw method erase the object before drawing it in a new location.

Change4: Add **checkKey()** and **if** statements that will change dx and dy so the circle's movement is controlled by the keypad.

This is what you should end up with:

```
from graphics import *
win = GraphWin("Output", 640, 480,1)

x=100          #Position
y=100
dx=0#Speed
dy=0
dd=1  #the rate of change of the rate of change,
#that is its acceleration.

delay=1000000  #Wait for humans to see
#Change this for your computer speed

while 1:

    c = Circle(Point(x,y), 10)  #Draw it
    c.setOutline('black')
    c.setFill('red')
    c.draw(win)

    for d in range(delay):pass  #wait
```

```
    c.undraw()

    x=x+dx    #  Plot new position
    y=y+dy

kb=win.checkKey()
if kb=="4":dx=dx-dd#Left  button
if kb=="6":dx=dx+dd#Right button
if kb=="8":dy=dy-dd#Up    button
if kb=="2":dy=dy+dd#Down  button

#Wrap if too far
if x>640 :x=x-640#Right
if x<1:x=x+640#Left
if y>480: y=y-480#Down
if y<1: y=y+480#up
```

Break Peddle Challenge:

Modify the code above so when the "5" button is pressed the circle comes to a complete halt.

Review: Moving More Than One Object

After our earlier lesson on making a circle move you were challenged to make two circles move. Here is an example of one possible solution to that problem:

```
from graphics import *

win = GraphWin("Output", 800, 600,1)
x1=100
y1=100
dx1=2
dy1=-1

x2=120
y2=180
dx2=-1
dy2=3

delay=500000

c1 = Circle(Point(x1,y1), 10)
c1.setOutline('black')
c1.setFill('red')
c1.draw(win)
```

```
c2 = Circle(Point(x2,y2), 8)
c2.setOutline('yellow')
c2.setFill('blue')
c2.draw(win)

while 1:
    x1=x1+dx1
    y1=y1+dy1

    x2=x2+dx2
    y2=y2+dy2

    if x1>800 or x1<1: dx1=-dx1
    if y1>600 or y1<1: dy1=-dy1

    if x2>800 or x2<1: dx2=-dx2
    if y2>600 or y2<1: dy2=-dy2

    c1.move(dx1,dy1)
    c2.move(dx2,dy2)

    for d in range(delay): pass
```

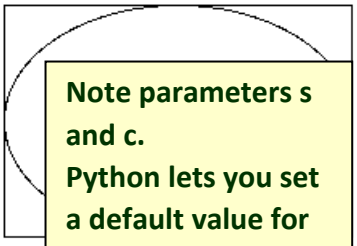
Remember that both objects needed to have their own location, speed, color and size, if they were going to behave differently. Using that principle, we are going to make a bomber and bomb that start off connected and later separate.

Making a Bomber Program

Making the Bomb

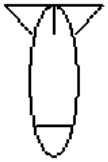
Zelle's graphics module includes an oval command. We are going to use it in the next program. This is how to use it:

```
object = Oval(Point(x1,y1),Point(x2,y2))
```



The point at **x1,y1** is the top left corner of a rectangle that contains the oval. Point **x2,y2** is the bottom corner.

Here is a function that makes a bomb out of an oval:



```
#Bomb function
from graphics import *
```

Note parameters s and c.
Python lets you set a default value for parameters by using the equal sign. They can be left out when the function is called. In this case size is assumed to be 20 with no fill color.

```
win=GraphWin("bomb",600,600)

#x,y=leading point,size, color
def bomb(x,y,s=20,c=""):
    global win, Oval, Line#From program
    global b1,b2,b3,b4.b5.b6 #Bomb objects

    w=int(s/2)#width from center
    h=w*3
    w=int(w/2)

    b1=Oval(Point(x-w,y),Point(x+w,y-h))
    if c!="":b1.setFill(c)
    b1.draw(win)   # body

    b2=Line(Point(x-w*0.8,y-h*0.2),Point(x+w*0.8,y-h*0.2))
    b2.draw(win)   # warhead line

    b3=Line(Point(x-w*2,y-h),Point(x+w*2,y-h))
    b3.draw(win)   # Horz line for fins

    b4=Line(Point(x-w*2,y-h),Point(x-w*0.8,y-h*0.8))
    b4.draw(win)   # Left fin

    b5=Line(Point(x+w*2,y-h),Point(x+w*0.8,y-h*0.8))
    b5.draw(win)   # Right fin

    b6=Line(Point(x,y-h),Point(x,y-h*0.8))
    b6.draw(win)   # Center fin
```

Try our new function out with the following function calls.

This call makes a standard size bomb with no fill with its nose at 200,300.

```
bomb (200,300,50)
```

This call makes a large red bomb with its nose touching 400,300.

```
bomb(400,300,80,"red")
```

Since we are going to animate the bombs, we will need to erase the bombs we make. Here is a function that will erase our bomb.

```
def unbomb():
    b1.undraw()
```

```
b2.undraw()
b3.undraw()
b4.undraw()
b5.undraw()
b6.undraw()
```

Bomb Challenge:

Make a function that draws a bomb at any location and of any size, but do not use the oval command.

Making Simple Explosions

Since we have a bomb it would be a good idea to make an explosion. Explosions can be one of the most difficult things to program. The physics of flying debris, flames, and smoke present mathematical problems. Here are some beginner ideas and tips

Tip1: Make the explosion part of the rest of the program so other objects continue to animate.
Tip2: Create a function that will place the explosion and set its size.
Tip3: Construct an aftermath image of the smoldering ruins.

Circle Explosion:

```
ground0=Circle(Point(bx,groundLevel),30)#Explosion
ground0.setFill("yellow")
ground0.setOutline("yellow")
ground0.draw(win)
for I in range(delay):pass
ground0.undraw()
```

Picture Explosion

To make a great looking explosion, you can simply show a picture of the explosion using Zelle's image command. Here is an example.

```
from graphics import *
win=GraphWin("",400,400)

ex=200
ey=200
er=50
delay=3000000
k=""
```

Here is the picture

```
while k!="space":
    k=win.checkKey()

ground0=Image(Point(200,200),"Explode1.gif")#Explosion
ground0.draw(win)
for I in range(delay):pass
ground0.undraw()
```

Python works with GIF pictures, not JPGs. It is a good idea to give the picture an invisible background. Here is a good cloud-based program: https://www298.lunapic.com/editor/?action=transparent .

Bounce Explosion

Here is an explosion that grows and shrinks using the same ideas we used to bounce a circle.

```
from graphics import *
win=GraphWin("",400,400)

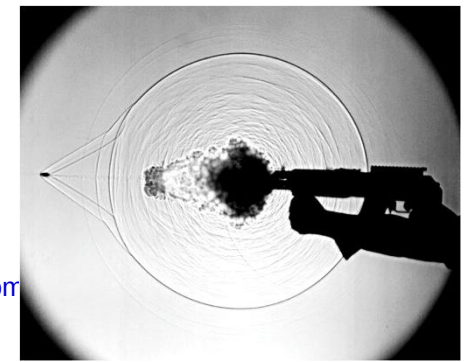
ex=200
ey=200
er=1
der=1#rate of radius size change
erMax=40
delay=30000
k=""

while k!="space":#Space bar sets off explosion
k=win.checkKey()

while er>0: #Stop explosion when it shrinks to zero size
    ground0=Circle(Point(ex,ey),er) #Draw it
    ground0.setFill("orange")
    ground0.setOutline("orange")
    ground0.draw(win)

    for d in range(delay):pass #Wait
    ground0.undraw()           #Erase it
er=er+der#Change its size
    if er>erMax : der=-der     #After max size shrink down
```

Shock Wave Explosion



Any sufficiently advanced technology is indistinguishable from

This build and decay explosion builds from the center, then eats itself away from the inside.

```
from graphics import *
win=GraphWin("",400,400)

ex=200
ey=200
er=1
der=1
erMax=30
delay=10000
k=""

while k!="space":
    k=win.checkKey()
    if k!="":print(k)

for er in range(erMax):#Expand explosion
    ground0=Circle(Point(ex,ey),er)
    ground0.setFill("red")
    ground0.setOutline("orange")
    ground0.draw(win)
    for d in range(delay):pass

for er in range(erMax+1):#Contract explosion
    ground02=Circle(Point(ex,ey),er)
    ground02.setFill("white smoke")
    ground02.setOutline("white smoke")
    ground02.draw(win)
    for d in range(delay):pass

ground0.undraw()
ground02.undraw()
```

Bomber Program

The example program below has two animated objects and reacts to a keypress. In addition, it applies what we learned about gravity, and has an integrated explosion. It uses our bomb functions. Try it out.

```
#Bomb Drop
from graphics import *  #  Bomb Program
win=GraphWin("bomb",600,400)
```

```
px=300#Plane
py=40
pr=10
dpx=-1
dpy=0

bx=px  #Bomb
by=py+pr
br=10
dbx=dpx
dby=dpy

g=1#gravity strength
delay=600000
k=""#keyboard buffer initial contents
drop=0#1 if dropping, 0 if with plane
dwell=3#How many times through the loopthe blast exists
dwellC=0#Dwell Counter
groundLevel=370#A line and the level bombs explode

ground=Line(Point(0,groundLevel),Point(win.width,groundLevel))
ground.draw(win)

def bomb(x,y,s=20,c=""):   #x,y=leading point,size
    global win, Oval, Line
    global b1,b2,b3,b4,b5,b6
    w=int(s/2)#width from center
    h=w*3
    w=int(w/2)

    b1=Oval(Point(x-w,y),Point(x+w,y-h))
    if c!="":b1.setFill(c)
    b1.draw(win)  # body

    b2=Line(Point(x-w*0.8,y-h*0.2),Point(x+w*0.8,y-h*0.2))
    b2.draw(win)  # warhead line

    b3=Line(Point(x-w*2,y-h),Point(x+w*2,y-h))
    b3.draw(win)  # Horz line for fins

    b4=Line(Point(x-w*2,y-h),Point(x-w*0.8,y-h*0.8))
    b4.draw(win)  # Left fin

    b5=Line(Point(x+w*2,y-h),Point(x+w*0.8,y-h*0.8))
    b5.draw(win)  # Right fin

    b6=Line(Point(x,y-h),Point(x,y-h*0.8))
```

```
    b6.draw(win)  # Center fin

    #Example1 - bomb (300,300,50)
    #Example2 - bomb(400,300,80,color_rgb(250,200,0))

def unBomb():
    b1.undraw()
    b2.undraw()
    b3.undraw()
    b4.undraw()
    b5.undraw()
    b6.undraw()

def plane(x,y):
    global PlaneObj
    PlaneObj=Circle(Point(x,y),10)# The plane is just a circle
    PlaneObj.draw(win)

def unPlane():
    PlaneObj.undraw()

#-------------  Main Program  ------------------
while k != "Escape":  #Press ESC to end program
    plane(px,py)#---Draw all objects
    bomb(bx,by,br)

    for d in range(delay):pass#---wait

    unPlane()#---Erase all objects
    unBomb()

    px=px+dpx#--- Move all objects
    py=py+dpy
    bx=bx+dbx
    by=by+dby
    if px<1 :#wrap the plane if it goes off the window
        px=px+win.width#win.width is the width in GraphWin
        bx=bx+win.width
    k=win.checkKey()
    if k=="space":# Press space to drop the bomb
        drop=1
    if drop==1:# if bomb dropped
        dby=dby+g
        if by>groundLevel:#if it hit ground
ground0=Circle(Point(bx,groundLevel),30)#Explosion
            ground0.setFill("yellow")
            ground0.setOutline("yellow")
```

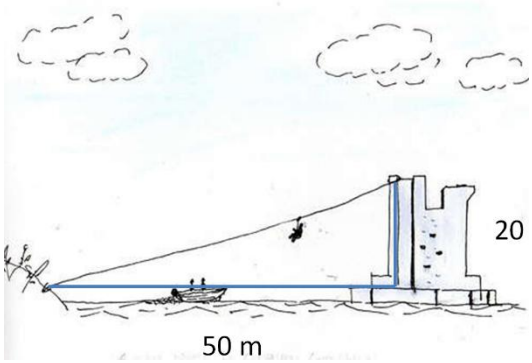
```
                ground0.draw(win)
                drop=0#Reset the bomb
                bx=px
                by=py+pr
                dby=0
                dwellC=dwell#how many loops explosion
        if dwellC>0: #Keep exploding while counting down
            dwellC=dwellC-1
            if dwell==0: ground0.undraw()#Blast goes away

win.close()  #----------- End of Program
```

Bomber Challenge:

Type in the bomber program and get it running, then modify it so the bomber looks like a plane and put a walking man on the ground below.

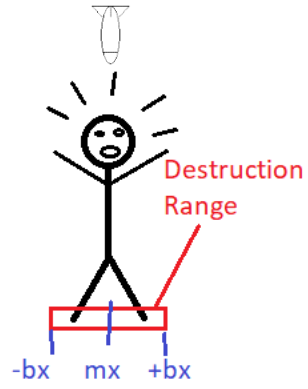
Proximity



If we are going to bomb things, the program is going to need to know if the bomb hit the target. Because computer screen images are made of thousands of dots[27] it ends up being rare that the position of one particular dot in one object will be exactly the same as one particular dot in another object.

In the case of ground based targets, like a walking man, the solution is easy. When the bomb detonates on the ground, calculate the distance the bomb is from the man. Do this by subtracting the horizontal position of the man, mx, from the horizontal position of the bomb, bx, and take the absolute value of the difference.[28] Here is a formula:

```
dist=abs(bx - mx)
```



Then use that distance in an **if** statement. If the distance is less than a certain amount, the bomb hit. Here is an example:

```
If dist<10:
```

Most games with explosions are more interesting if the damage done by the bomb depends upon how close the bomb is to its intended target. Use

[27] The dots on the screen that make up the image are called **pixels**, short for picture element.

[28] Remember that absolute value removes the negative sign from any number, so abs(-5) is equal to 5. Math people define it as a number's distance from zero, which makes it perfect for distance calculations.

the distance you calculated earlier to calculate the damage the target took.

According to a scientist named Newton, the damage done will equal the power of the explosion divided by its distance from the target squared. Here is an example:

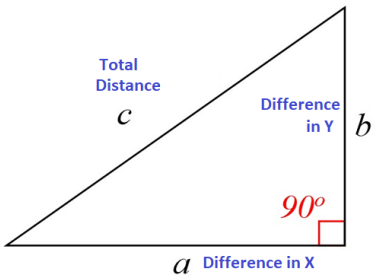
```
health=health - power/(dist^2)
```
[29]

Self-Destruct Challenge:

Make a circle travel across the screen. Make itdetonate when it gets within 10of the right edge of the window.

2-D Proximity

One way of calculating distance when objects are at different heights is by using the Pythagorean Theorem.[30] Once again the ancient Greeks figured out how to solve the problems we encounter while creating advanced technology. The square of the distance is equal to the square of how far apart theobjects are vertically plus the square of how far apart the objects are horizontally.



$$c^2 = a^2 + b^2$$

$$\sqrt{c^2} = \sqrt{a^2 + b^2}$$

$c = \sqrt{a^2 + b^2}$ = distance

But, a is how far apart in **x**

And b is how far apart in **y**, so…

$$dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Here is an example in python that calculates the distance between an object at (x1,y1) and a second object at (x2,y2):

```
from math import sqrt# Get the square root function
dist = sqr( (x1-x2)*2 + (y1-y2)*2 )
```

The square root function is time consuming for the computer. Here is another way to calculate the distance using trig that gets rid of the square root and both squares:

```
from math import atan, sin# Get the trig functions
dist = (y2-y1)/sin(atan((y2-y1)/(x2-x1)))
```
[31]

[29] The parentheses are unnecessary in this formula because raising numbers to powers has operation precedence over division, as it does in math.

[30] Philosopher and mathematician, Pythagoras lived about 2500 years ago. He discovered that in a right angled triangle: the square of the hypotenuse is equal to the sum of the squares of the other two sides. The hypotenuse is the longest side.

[31] D. J. Bouwsma 2018.

Speed Challenge

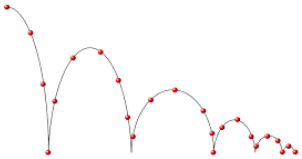
In animation, execution speed is key. Fast programs produce smooth attractive graphical effects. Write a program to find out which of the two distance calculations is faster. Use the timer() function below to capture the system time in millionths of seconds to store the time before the test and after the test. This is how to import it:

```
from timeit import default_timer as timer
```

Capture the time like this:

```
beginTime = timer()
```

Friction and Elasticity



In the real world, basket balls stop bouncing. In the real world, a rolling ball does not slow down forever, it stops.

Every time an object impacts, a portion of its energy is lost. This energy, according to scientists, is turned into heat.[32]The amount of energy retained to bounce is called elasticity.

Programming elasticity is easy. Each time a circle impacts and reverses vertical motion, multiply by the object's elasticity constant. The elasticity constant varies for different objects, but it is always less than one, a perfect bounce. A super ball might have an elasticity of 0.95, while a basketball may be closer to about 0.75. Here is what the bounce command would look like.

```
e=.6#  Elasticity Basket Ball← At the top of the program
…
    if y > 600 - r or y < r: dy=-dy*e # Elasticity of bounce
```

As a basketball rolls friction constantly slows it down. Here is what friction might look like in a Python program.

[32] All forms of energy, in the real world, slowly degenerate into heat. Heat, or **Brownian Motion**, as it is called by scientists, is a measure of how fast molecules are vibrating. This degeneration to heat is called **entropy**, and it is the second **law of thermal dynamics**. According to this law, the world is a fallen place that naturally tends to toward disorder. Any organization in the world requires energy from outside itself, creating even greater disorder outside it. So, bouncing basket balls and humans with complex bodies exist by increasing the disorder in the world.

```
f=.992#  Rolling Friction ball
```
← **At the top of the program**

…

```
if y==600-r:dx=dx*f# Forward motion friction
```

Mathematically, a rolling ball will continue to roll forever and a bouncing ball will continue to bounce at lower heights forever, but a second form of friction brings objects to a complete halt. Scientists believe there are quantum level chemical bonds forming between moving objects and surfaces that finally bring objects to a complete stop.

Use an if statement to stop movement completely when it gets too small. Here is what that might look like in a program. Because motion can be positive or negative, it is a good idea to use the absolute value function abs(). It will make negative motions positive, so we can check their strength apart from their direction. This is what that might look like.

```
if abs(dx)<.05:dx=0# Stopping friction
```

Here is a complete example of a program that simulates a bouncing object.

```
from graphics import *
from math import *

win = GraphWin("Output", 800, 600,1)
x=100
y=100
r=10
dx=1.0
dy=0.0
d2=0
g=.05
delay=300000

#e=.85#  <---- Elasticity Super Ball
e=.6# <---- Elasticity Basket Ball
#e=.3  # <---- Elasticty  Human Body

f=.992#  <---- Rolling Friction ball

while 1:
    c = Circle(Point(x,y), r)
    c.draw(win)
    c.setWidth(3)

    for d in range(delay):pass

    dy=dy+g  #add gravity
```

```
x=x+dx
y=y+dy
if x>800-r or x<r: dx=-dx

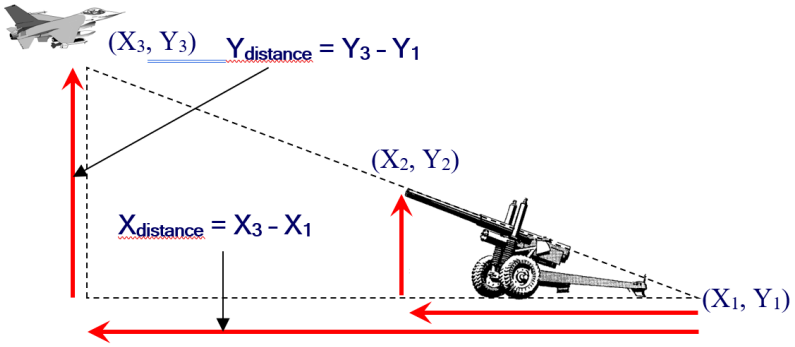
if y>600-r or y<r:

    dy=-dy*e# Elasticity of bounce
    y=y+dy
    if abs(dy)<.15:# Stopping friction
        dy=0
        y=600-r  #Fix any fractions
        dx=dx*f# Forward motion friction

if y==600-r:dx=dx*f# Stop Forward motion friction
if abs(dx)<.05:dx=0# Stopping friction

c.undraw()
```

Autotargeting



You can program a gun to aim wherever the mouse points or wherever an airplane flies. Using the same principles, you can make an enemy always seek a player. All it takes is a little geometry. If you look at the picture carefully, you will notice that the triangle to the plane is the same as the triangle made by the gun.[33] That means the ratios are the same in both triangles.

For example, if the gun is 5 in length, and the end of it 3 in height then if the plane is three times further away, a distance of 15, then the height of the plane must be three times more as well, making it 9.

To keep the Xs and Ys proportional, simply, multiply the ratio of the length of the gun and the distance of the object by X and Y.This code calculates the horizontal location of the end of a gun barrel.

```
GunEndX = GunLength * RatioXtoTotal + GunBaseX
```

To calculate the distance from the gun to the object, use the distance formula we studied earlier. (Click here to review it.)

[33] The two triangles that both start at (x1, y1) are called similar triangles by math people. They are essentially the same triangle resized.

To calculate the vertical and horizontal distance the object is from the gun do not forget to take the position of the gun into account, and do not forget that vertical numbers move down a computer screen. Here is an example.

```
obDistX = GunBaseX-ObjectX
```

Here is an example of a complete autotargeting program with a gun tracking an animated circle.

```
#Autotargeting
from graphics import *
from math import sqrt# for distance formula
win=GraphWin("",400,400)

def dist(x1,y1,x2,y2):
return sqrt((x2-x1)**2+(y2-y1)**2)


x=200
y=180
r=5
dx=1

delay=100000

gr=40  #Inital position of gun
gx=200
gy=300-gr #Points straight up

gnd=Line(Point(0,300),Point(400,300))#The ground
gnd.draw(win)

while 1:
    c=Circle(Point(x,y),r) # Draw circle and gun
    c.draw(win)
gun=Line(Point(200,300),Point(gx,gy))
gun.draw(win)

    for d in range(delay):pass # wait

    c.undraw() #Erase circle and gun
    gun.undraw()

    x=x+dx #  Move circle
    if x>400:x=x-400
```

```
cr=dist(x,y,200,300)# Gun(200,300)distance from circle

ratioX=-(200-x)/cr# distance x from gun / total distance
gx=gr*ratioX+200#Gun length * ratio + gun base location

ratioY=(300-y)/cr# distance y from gun / total distance
gy=300-gr*ratioY#Gun length * ratio + gun base location
```

Mouse In Real Time

The Zelle graphics includes a way to access the mouse here is how.

```
m =win.checkMouse()
```

When the user clicks, Python sets variable m to a point that contains the x, y coordinates. In a real time program, you will need to check the mouse and retrieve the location that was clicked. Here is code that would check for a mouse click and store the location the user clicked.

```
if m != None:  # This check for anything
        mx = m.getX()
        my = m.getY()
```

When m is not equal to None, a special variable that means nothing, the methods, getX and getY will contain the location of the mouse click. Here is a program that drops bombs from the location a person clicks. It uses the bomb functions we wrote earlier.


```
#  Bomb Drop from Mouse
from graphics import *
win=GraphWin("bomb",600,600)

def bomb(x,y,s=20,c=""):   #x,y=leading point,size
    global b1,b2,b3,b4,b5,b6
    w=int(s/2)#width from center
    h=w*3
    w=int(w/2)

    b1=Oval(Point(x-w,y),Point(x+w,y-h))
    if c!="":b1.setFill(c)
    b1.draw(win)  # body

    b2=Line(Point(x-w*0.8,y-h*0.2),Point(x+w*0.8,y-h*0.2))
    b2.draw(win)  # warhead line

    b3=Line(Point(x-w*2,y-h),Point(x+w*2,y-h))
```

```
    b3.draw(win)  # Horz line for fins

    b4=Line(Point(x-w*2,y-h),Point(x-w*0.8,y-h*0.8))
    b4.draw(win)  # Left fin

    b5=Line(Point(x+w*2,y-h),Point(x+w*0.8,y-h*0.8))
    b5.draw(win)  # Right fin

    b6=Line(Point(x,y-h),Point(x,y-h*0.8))
    b6.draw(win)  # Center fin

    #Example1 - bomb (300,300,50)
    #Example2 - bomb(400,300,80,color_rgb(250,200,0))

def unbomb():
    global b1,b2,b3,b4,b5,b6
    b1.undraw()
    b2.undraw()
    b3.undraw()
    b4.undraw()
    b5.undraw()
    b6.undraw()

#========= Main Program =========
bx=0
by=0
bdx=0
bdy=0
be=0
g=.5

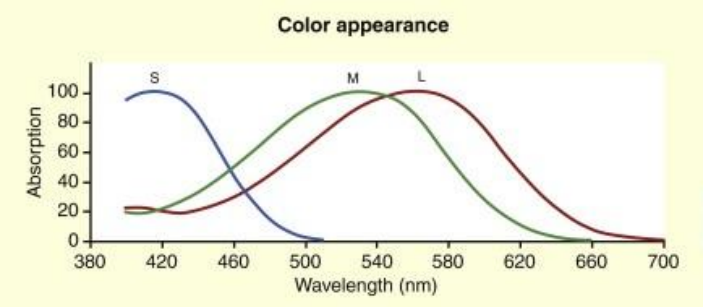
while 1:
    mouse=win.checkMouse() #Try to get mouse location
    if mouse != None and be==0:#if clicked and bomb not
                               #dropping already.
        bx=mouse.getX()
        by=mouse.getY()
        bdx=0
        bdy=0
        be=1
    if be==1:#if bomb exists
        bomb(bx,by)
        bdy=bdy+g  #add gravity
        bx=bx+bdx   #move bomb
        by=by+bdy
        print(by)
```

```
for d in range(1000000):pass#Wait

unbomb()#Erase

if by>500:  #if it hit the ground
    be=0
    expl=Circle(Point(bx,500),30)# Explosion----
    expl.setFill("yellow")
    expl.draw(win)
    for e in range(500000):pass #Wait
    expl.setFill("orange")
    for e in range(500000):pass #Wait
    expl.setFill("red")
    for e in range(500000):pass #Wait
    expl.undraw()
```

Mixing Colors



Python is capable of producing over 16 thousand different colors and shades. Most scientists believe the human eye can see over 10 million different colors, but we, as humans, summarize the information coming in, and can only distinguish between about 300 colors when we are not concentrating on the differences. The eye has three light sensors. **Rods** that will detect only black and white, but operate well at night, and **cones**. The cones come in two types[34] that see colors that oppose eachother. There are cones that see red or green, and cones that see blue or yellow.

Computer monitors and TVs mix red, green, and blue to make all colors.Here is an on-line mixing program http://www.csfieldguide.org.nz/en/interactives/rgb-mixer/index.html. Try mixing green and red, and see what you get.

Python uses an HTML command to mix colors. In HTML, a color is specified by placing a pound sign in front of three two digit numbers, all in quotes. See the box to the right to understand how it works.Here are two examples.

mixedColor = "#RRGGBB"
RR – intensity of red in hex
GG – intensity of green in hex
BB – intensity of blue in hex
Hex is base 16 – count to 16 before putting 1 in the "tens" place.

[34] Some people called tetrachromats have a rare gene that allows them to see colors in the ultraviolet spectrum. When asked to describe the color of mowed grass, one tetrachromat described it this way: I see pinks, reds, oranges, gold in the blades and the tips, and gray-blues and violets and dark greens, browns and emeralds and viridians, limes and many more colors. Click this https://xritephoto.com/cool-tools to see if you are a tetrachromat.

```
turtle.color("#ff9900")#Orange
circle2.setFill("#ccccff")#Light blue
```

The numbers are hexadecimal numbers. In **Hexadecimal** or **hex**, here are 16 different digits rather than 10. The first ten use the same digits 0 – 9 that **decimal**[35] uses after that letters A through F are used.

Hexadecimal:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

So the number 21 be 16 + 5 and look like **15**. The number 300 would look like 256 (16^2x1) + 32 (16x2) + 12 which would be written **12c**.[36]

Color Mix Challenge:

Write a program that draws two circles. One with fill the color of your hair, the other filled with the color of your shirt.

Bouncing Color and Hue

Here is a function that converts three decimal numbers into an HTML style color:

```
def rgb(r,g,b): #enter: Red, Green, Blue 0-255
        return '#%02x'%r+'%02x'%g+'%02x'%b #HTML color string
```

The % in each string above is substituted with the r, g, and b that follow them, that is, the amount of red, green, and blue.

This program bounces a circle's blue color the way we bounced a circle earlier.

```
from graphics import *
win=GraphWin("Bouncing Colors",400,400)

def rgb(r,g,b): #enter: Red, Green, Blue 0-255
        return '#%02x'%r+'%02x'%g+'%02x'%b#HTML color string[37]

c=100    #Initial level of color (0 to 255)
```

[35] Decimal is the numbering system all of us use normally. Scientists believe that we count to ten before grouping numbers because we have ten fingers.

[36] See https://www.mathsisfun.com/hexadecimals.html for a full explanation of how hex works.

[37] The % is substituted for the values of r, g, and b. The zero in 02x specifies preceding zeros; the two the length of the number, and the x that it will be in "hex".

```
dc=1    #Color change rate, works the way dx works
cir=Circle(Point(200,200),150)
cir.setOutline("black")
cir.draw(win)

while 1:
c=c+dc# Change the color intensity
if c>254 or c<1:   dc=-dc# Bounce the color change rate
cir.setFill(rgb(0,0,c))

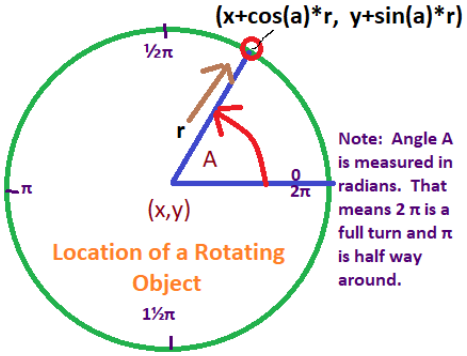
    for d in range(80000):pass
```

Color Bounce Challenge:

1. Write a program that bounces of a red, green and blue circle, so all three fade their colors at once.
2. Write a program that makes a circle that pulses in all three colors at once, but have each color bounce at a different rate, so the colors blend.

Spin Using Sin and Cos

Radar



Ancient Greek math experts figured out how to calculate the position of turning objects and invented trigonometry.[38]Trigonometry works using two math functions: Sine, abbreviated **sin**, and Cosine, abbreviated **cos**. For a spinning object, sin tells you how much to go up and down, and cos tells you how much to go back and forth. In math sin gives you **y**, and cos gives you **x**. Here are two things to remember when you program spinning objects:

1. **Add the original position**, otherwise your object will circle the upper left corner.
2. **Multiply by the radius** (how far it is from the point it circles) or it will make such small circles thatyou will not be able to see it move.

[38]The term "trigonometry" was derived from Greek τρίγωνον trigōnon, "triangle" and μέτρον metron, "measure". The modern word "sine" is derived from the Latin word sinus, which means "bay", "bosom" or "fold." https://en.wikipedia.org/wiki/History_of_trigonometry

```
#Radar

from graphics import *
from math import *

win = GraphWin("Output", 800, 600,1)
delay=10000

a=0.0
while 1:
#--------Draw object
ln=Line(Point(400,300),Point(400+cos(a)*200, 300+sin(a)*200))
    ln.draw(win)

#--------wait
    for i in range(delay): Pass

#--------erase object
    ln.undraw()

#--------move object
    a=a+.01
    if a>3.14159*2:# if it went all the way around
        a=a-3.14159*2# subtract a full turn
```

Spiral

By increasing both the angle and the radius, sin() and cos() will produce spirals. Try this program:

```
from graphics import *
from math import sin,cos
can=GraphWin("Spiral",400,400)

x=200
y=200
r=5
dr=0.3
a=0
da=.05

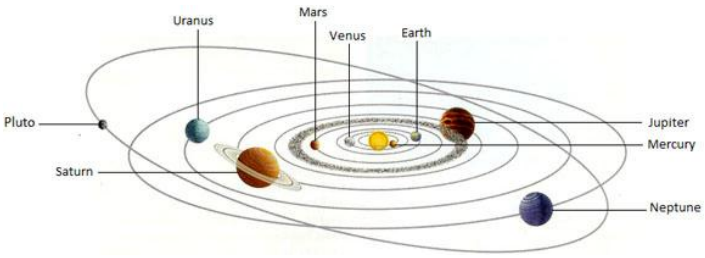
for i in range(600):
a=a+da
    if a>3.14159*2:a=a-3.14159*2
r=r+dr
    c=Circle(Point(x+cos(a)*r,y+sin(a)*r),5)
```

Try da=6

```
c.draw(can)
```

Solar System Challenge+:

Use sin and cos to make a solar system animation.**HINT: LOOK UP KEPLER'S LAW.**



Center of Gravity

Our radar example works for a spinning object with no mass[39], like the light from a spinning flashlight, but almost everything around us has weight and does not spin from one of its far edges. If an object is perfectly shaped, it will spin at its center, and its movement is calculated from its center point, like a circle.

Here is an example of a bouncing bar that rotates around its center of gravity:

```
#Spinning Bouncing Bar
from graphics import *
from math import *
win = GraphWin("Output", 800, 600,1)

delay=30000

x=win.width/2
y=win.height/2
r=40
a=0.0
dx=1# Horizontal Speed
dy=1# Vertical Speed
da=.01# Spin Speed

while 1:
#--------- Draw object
    x1=x-cos(a)*r  # Plot object remember:
    y1=y-sin(a)*r  #      sin is how much of
#      the angle is up and
    x2=x+cos(a)*r  #      down and cos is how
    y2=y+sin(a)*r  #      much is right and left

    ln=Line(Point(x1,y1),Point(x2,y2))
```

[39] Mass (symbolized *m*) is a dimensionless quantity representing the amount of matter in a particle or object. The standard unit of mass in the International System (SI) is the kilogram (kg). from: https://whatis.techtarget.com/definition/mass-m

```
    ln.draw(win)
    ln.setWidth(6)

#--------- Wait
    for i in range(delay):pass

#--------- Erase object
    ln.undraw()

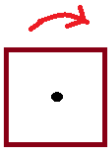
#--------- Move object
    a=a+da# Spin it
    if a>3.14159*2:
        a=a-3.14159*2

    x=x+dx# Move it
    y=y+dy

    if x>win.width or  x<1: dx=-dx#Bounce
    if y>win.height or y<1:dy=-dy
```

Spinning Square Challenge:

Rotate a square around its center point.



Angled Projectiles

We made a bar spin using sine and cosine, sending objects on an angled path uses the same math, except when calculating dx and dy, multiply by the force rather than the distance from the point the object is pivoting on. Like this.

```
force = 1.5       #Figure the force out by what works
a = 3.14159/4     #a 45 degree angle for example[40]
dx = cos(a) * force  #portion of speed that is sideways
dy = -sin(a) * force #portion of speed that is up and down
```

Here is a gun program that uses sine and cosine to aim the gun barrel and give the bullet an angled trajectory that is the same as the angle on the gun. I uses the checkKey() event function discussed earlier. Click here if you want to learn about the checkKey() function and real time programming. Click here if you want a refresher on how gravity works.

```
# Gun
from graphics import *
win=GraphWin("gun",1024,768)
```

[40] Reversing direction is π, or 180 degrees, so 90 degrees is ½ π and 45 degrees is ¼ π.

```
from math import sin, cos, pi

delay=500000
power=3      #gun power
g=.02          #gravity
#-----------------------------------

x=win.width/2#Position and size of gun
y=win.height*.9
r=10
gr=r*2# Gun barrel radius
a=-pi/2 #Straight up (negative because monitor increases down)
da=.1

gx=0.0#gun barrel, (decimals will allow fractions)
gy=0.0

bx=0.0#bullet info
by=0.0
br=5
bdx=0.0
bdy=0.0
be=0#bullet exists switch →(1=true, 0=false)

turret=Circle(Point(x,y),r)
turret.setFill("black")
turret.draw(win)
ground=Rectangle(Point(0,y), Point(win.width,win.height))
ground.setFill("#803000")#A nice brown
ground.draw(win)

while 1:
 #===== Draw all objects ======
    barrel=Line(Point(x,y),Point(x+cos(a)*gr, y+sin(a)*gr))
    barrel.setWidth(3)
    barrel.draw(win)

if be==1:#Bullet is in flight, so draw it
        bullet=Circle(Point(bx,by),br)
        bullet.setFill("blue")
        bullet.draw(win)

 #=====  Wait  ======
    for d in range(delay):pass

 #===== Check Keyboard ======
    k=win.checkKey()#Check for keypress
```

win.width finds the width
win.height finds the height

```
    if k=="4": a=a-da #angle gun according to command
    if k=="6": a=a+da

    if k=="space" and be==0:#Fire ! if we have not already
bx=x+cos(a)*gr#set bullet initial position
by=y+sin(a)*gr
dbx=cos(a)*power#set bullet initial movement
dby=sin(a)*power
be=1#Make bullet exist
        bullet=Circle(Point(bx,by),br)
        bullet.draw(win)


 #===== Erase Objects======
    barrel.undraw()
if be==1:# If bullet exists, erase and move it
        bullet.undraw()
        dby=dby+g#  Add in gravity
        bx=bx+dbx#  Move bullet
        by=by+dby
        if bx>win.width or bx<0: be=0#off screen
        if by>y:# --hit ground explosion here?
be=0# Bullet no longer exists
```

Multiple Objects

Earlier we programmed two bouncing circles. Remember that both circles needed their own x, y, dx and dy. In the program we named them x1,y1, dx1, dy1, x2,y2, dx2, and dy2. Using the principles, we would need three sets of each variable to make three circles and 50 sets of each variable to make 50 circles.

Python has a quicker ways of handling all those variables. One of the best ways to handle multiple objects is to use a list. A list can hold hundreds, even thousands of objects and let you go through them using numbers.

The simplest way to make a list is just to set a variable equal to a bunch of items **separated by commas in square brackets**.

Try this program out. It creates a list of numbers:

```
x=[20,30,40,50]
print(x)
```

This one creates a list of words – in this case, colors used in elecronics:

```
band=["black","brown","red","orange","yellow",
      "green","blue", "violet","grey","white"]
print(band)
```

[41]

You can get individual items from the list by specifying which one you want using a number in brackets. There is a trick to this, however. Python starts **counting from zero**, so zero, in our previous list, is "black" and two is "red".

Replace the print command in the previous program with this, and try the program again.

```
print(band[0], band[2])# This should print black and red
```

The numbers in the brackets are called a variable's **index** by programmers and math people.[42]Indexes are just numbers to Python. You can generate them using math, **for** loops, math, or just simply type inthe number you want like we did in the previous example.

Here is an example of for loop that prints all the colors in the list along with the number they represent:

```
band=["black","brown","red","orange","yellow",
      "green","blue", "violet","grey","white"]
for i in range(10):
    print(i, band[i])
```

Try this example that uses a list to change colors by number with turtle graphics.

```
import turtle
win=turtle.Screen()
t=turtle.Turtle()
t.speed(0)

band=["black","brown","red","orange","yellow",
      "green","blue", "violet","grey","white"]

t.forward(-280)
for c in range(10):

t.color(band[c])
```

[41] These colors are the color codes for the rings, or bands, on resistors. They tell engineers how strong the resistor is.

[42] In math, indexes are used in summations that use a big capital sigma that looks like this: ∑ Perhaps you have seen these in your math book. By custom, the letter **i** is used for the index.

```
for i in range (30):
    t.left(90)
    t.forward(200)
    t.backward(200)
    t.right(90)
    t.forward(2)
```

Some Helpful List Methods[43]

list.append(item) - adds an item to the end of the list.

list.insert(index, item) - inserts the element at the given index, shifting all the other items in the list to the right.

list.index(item) - searches for the item and returns its index.

list.remove(item) - searches for the first instance of the given element and removes it

list.sort() - sorts the list

list.reverse() - reverses the list in place

list.pop(index) -- removes and returns the item at the given index. Returns the

Circles with Lists

Lists can have no items in them. To make a list with no items, set a variable equal to two brackets with nothing in them, like this. This is very handy when you want the computer to make the list using **append**.

```
v = []
```

Here is a program that makes 50 circles at random locations using the random number generator we studied earlier, empty lists, and the **append** method from the box above.

```
from graphics import *
from random import random as rnd#As give it a short name

win = GraphWin("Output", 800, 600)

n=50# How many circles
x=[]
y=[]
c=[]# List of circle objects
```

[43] Derived from: https://developers.google.com/edu/python/lists

```
for i in range(n):
x.append(int(rnd()*600)+100)
y.append(int(rnd()*400)+100)
c.append(Circle(Point(x[i],y[i]),5))
    c[i].setFill('red')
    c[i].draw(win)
```

Random Circle Challenge:

Modify the program above so there are more circles and they are random colors and sizes.

Multiple Object Animation

By using lists to contain not only the x and y, but also the dx and dy values it is easy to make a program that animates many circles at once. In the previous program, we made random circles.This program moves them.

```
#Bounce Circles in a List
from graphics import *
from random import random as rnd #As gives it a short name

win = GraphWin("Output", 800, 600,1)

n=50#How many

x=[]#Set up empty lists
y=[]
dx=[]
dy=[]
c=[]#Circle objects

delay=1000  #Can be set to zero
ob=0#Objects take turns -- its this one's turn

for i in range(n):  #Initial postion and speed
    x.append(int(rnd()*600)+100)
    y.append(int(rnd()*400)+100)
    dx.append(rnd()*6+.3)
    dy.append(rnd()*6+.3)
    if rnd()<0.5:dx[i]=-dx[i]  # Half the time go other way
    if rnd()<0.5:dy[i]=-dy[i]
    c.append(Circle(Point(x[i],y[i]),5))
    c[i].setFill('red')
    c[i].draw(win)
```

```
while 1:
ob=ob+1# Do next object
if ob>n-1:ob=0# go through again

    c[ob].undraw()      #--- Erase ---
x[ob]=x[ob]+dx[ob]#--- Move ---
y[ob]=y[ob]+dy[ob]

    for i in range(delay):pass

    c[ob]=Circle(Point(x[ob],y[ob]),5)#--- Draw ---
    c[ob].setFill('red')
    c[ob].draw(win)
```

List Bounce Challenge:

Modify the program above so all the circles bounce.

Explosions with Trig

To make an explosion that blows an object to bits, you need the following:

1. **A maximum for the explosion energy** – in this program **power**, located at the top of the program
2. **The number of parts (bits)** – in this program **n**, located at the top of the program.
3. **A direction for each part** – in this program:**dx=cos(a)**, **dy=sin(a)** where **a** is a random angle 0 to 2π. Cos() will give you the portion of the movement that is horizontal; sin() will give you the portion that is vertical.[44] All you need is the angle.
4. **Individual part power** made up of a random amount of the maximum – in this program 50% of the maximum energy is applied to all the parts and a random amount of the other 50% is added in.
5. **An initial location** – in this program, (**xg0, yg0**) is the location of ground zero.

```
# Explosion using lists and trig
# (Blowing things to bits)
from graphics import *
from math import sin, cos
from random import random
```

[44] Math experts call the horizontal position the **x**axis and the vertical position the **y**axis. An axis is one of two measures used to make a location. In the case of a computer, the x axis measures how far the location is from the left edge of the window, and the y axis measures how far it is from the top. Dx is how fast the object is moving away from the left. If it is negative, the object is moving toward the left edge of the window. Dy is how fast the object is moving down. Once again, if it is negative, it is moving up. Remember, on a computer, the y axis is upside down. In math, bigger y numbers mean the location is higher.

```
win = GraphWin("Output", 500,500)
win.setBackground("black")

xg0=win.width/2# Ground zero location, the center
yg0=win.height/2
power=4          # Max energy of explosion

n=50  #Number of bits in explosion

x=[]  #Set up empty lists
y=[]
dx=[]
dy=[]
p=[]  #Point objects (dots) for flying parts

delay=0  #Can be set to zero
ob=0  #Object number to manipulate

for i in range(n):  #---Initial postion and speed
    x.append(xg0)# Initial location
    y.append(yg0)
a=random()*3.14159*2#Make angle part flies in
v=power*random()*.9+.1*power#Velocity of part,90% random
dx.append(cos(a)*v)
dy.append(sin(a)*v)
    p.append(Point(x[i],y[i]))
    p[i].setFill('yellow')
    p[i].draw(win)

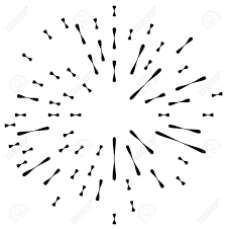
#Stop program until user clicks
go=Text(Point(win.height/2,40),"Click to Detonate")
go.setTextColor("orange")
go.draw(win)
mclick=win.getMouse()

while 1:
    ob=ob+1             # Do next object
    if ob>n-1:ob=0      # go through again

    p[ob].undraw()      #--- Erase ---
    x[ob]=x[ob]+dx[ob] #--- Move ---
    y[ob]=y[ob]+dy[ob]

    for i in range(delay):pass

    p[ob]=Point(x[ob],y[ob])#--- Draw ---
```



```
p[ob].setFill('yellow')
p[ob].draw(win)
```

Gravity on Parts Challenge

What goes up must go down. Add gravity to the example above. Click here to review gravity.

Time

According to Einstein's theories, has some strange properties. Fast objects age more slowly and change shape, and, according to Einstein, there is no reason for time to progress at a constant rate of speed, or to move forward at all. Strangely, if time stopped or even went backwards, we would never know it.

Python has a wide variety of time functions. We will discuss just a few.

Date and Time

Try this program out. It accesses the computer's internal clock to provide both the date and the time of day.

```
#Time of day
import time
tm=time.localtime(time.time())# This get the time
yr=tm[0]# tm is a list
mo=tm[1]
day=tm[2]
hr=tm[3]
min=tm[4]
sec=tm[5]
print(mo,"/",day,"/",yr,"  ", hr,":",min,":",sec)
```

Notice the output of print command correctly prints the date and time, but it inserts a space where ever there is a comma. Python version 3, the one you are using, provides a way to replace the space with anything you wish, including nothing at all, if you wish. Here is a better version of the print command above.

```
print (mo,day,yr ,sep="/", end="  ")
print (hr,min,yr, sep=":")
```

,sep="string" in a print command replaces commas with what ever is between the quotes.

Timer

Python also provides a way to measure elapsed time to within several millionths of a second.


```
# Time Elapse Program

from time import time

start=time()
for i in range(1000000):pass
end=time()
print("It takes ",end-start,"seconds to count to one
million")
```

Run the program more than once. The first time the program runs Python has to convert it into machine code. In addition, modern computers are run many programs in the background all the time. These programs interrupt Python, adding to the time elapsed.

Timer Challenge:

Write a program that tests the computer to see how many times it needs to loop to wait 1/60 of a second and print the result. Remember that the true time will be the lowest scores from multiple trials

This program measures how fast your reactions are. Try it and see how fast you are.

```
# Reaction time
from graphics import *
from time import time

win=GraphWin("",600,400)

txt=Text(Point(300,50),"When I say go press a key.")
txt.draw(win)

begin=time()# Wait several seconds to build tention
while time()<begin+2:pass#This line waits 2 seconds
for i in range(1,4):
    begin=time()
    txt=Text(Point(300,50+i*25),"Wait ...")
    txt.draw(win)
    while time()<begin+2:pass

txt=Text(Point(300,50+100),"G  o  !")
txt.draw(win)
begin=time()
k=""
while k=="":
    k=win.checkKey()
```

```
TotalTime=time()-begin# Calculate time since begin

txt=Text(Point(300,200),"Total time =" + str(TotalTime) +
"seconds.")
txt.draw(win)
```

Quiz Challenge

Write a program to quiz people in arithmetic. Measure their speed and accuracy.

Clock Movement

Using Python's **time** module you can make an analog clock. Analog clocks have hands that sweep across a clock face. Click here if you forgot how the time module works.

Clock hands spin with 60 locations per rotation, but in mathematics, a full spin is 2 times pi. So, to make a clock hand spin using the system time, you will have to use ratios to change the portion of the spin to an angle that sin() and cos() can read. If ***Angle sec*** is the angle of the second hand, the math looks like this;

$$\frac{Time\ in\ sec}{60} = \frac{Angle\ sec}{2\pi} \text{ so ... } \quad Angle\ sec = \frac{Time\ in\ \text{sec} \times 2\pi}{60}$$

Here is an example

```
#Second hand - seconds.py
from graphics import *
from math import sin, cos, pi
import time

win=GraphWin("second hand",800,600)

cx=win.width/2
cy=win.height/2
a=0.0
r=win.height*.4#Length of the hand is 40% of screen width
oldS=0

while 1:
t=time.localtime(time.time())
sec=t[5]
a=sec*2*pi/60#Change 60 rotation to 2pi[45]
    s=Line(Point(cx,cy),Point(cx+cos(a)*r,cy+sin(a)*r))
```

Time List Contents
[0] -> year
[1] -> month
[2] -> day
[3] -> hour
[4] -> minute
[5] -> second

[45] This could be reduced to sec * pi /30.


```
s.setOutline("black")
if s!=oldS:s.draw(win)#Draw it whan it changes
oldS=s

for i in range(400000):pass

if s==oldS:s.undraw()
```

Clock Challenge

Add an hour hand and a minute hand to the program above.

Strings

String Basics

In the late 1940s and 1950s, many people believed computers would become increasingly less important because nearly all the tables and numbers scientists use commonly would be calculated and printed. As it turns out, a computer's ability to store, manipulate and retrieve alphabetic information is more important than its ability to crunch math.

Strings are like lists, but they are lists made up of letters, symbols and numbers called **characters**. Here are some examples of characters. There are 256 different types of characters.[46]

```
W r $ [ 2 &z ? G ° π F ≥Æ
```

Characters are stored differently than numbers are. The smallest unit of memory is called a **byte**. One byte can store only one character or a number from -32,768 to +32,767.The reason so little information is stored is because there are so many different types of characters. Python will give you an error message, if you try to do mathematics with characters.

Creating and Storing Strings

Strings that cannot be changed are called string literals. You have actually created string literals before, but just did not know the name for them. To create a sting literal, simply put quotes (or apostrophes) around the characters you want in the string. Here are some examples.

[46] There are several standard codes that have been used to store letters. The one we are referring to is ASCII, American Standard Code International Institute. A newer code, called Unicode, uses much more memory, but allows multiple languages.


```
"AaBbCcDdEe" # Capitals have a different code than
'2 + 2 = 4'#                                small letters.
"A wise man overlooks an insult."
```

To create a string variable, simply set a variable equal to a string constant. Here are some examples.

```
a="Aahchoo"
b='God'
c="bless"
d=' you'#Notice the space before the y
```

Concatenation

In Python, you can tack strings together like railway cars in a train by using a plus sign. This is called concatenation. The plus sign is Python's concatenation operator. Here is an example that uses the code above.

```
e=a+b+c+d  #This code prints
print (e)  # AahchooGodblessyou
```
← Think about spacing in strings[47]

Here is another example.

```
print("2" + "2" )#This prints 22, not 4
```

Repeating Strings

Python will duplicate strings followed by an * and a number. The number tell Python how many copies to make. Try this program out.

```
toomany="x"*20
print(toomany)

n="N, N, capital N, we love you so much we'll say it again."
print (n*5)
```

N Challenge:

In Python "\n" (backslash n) is replaced by ENTER throughout strings.
Rewrite the program above so it prints the following text 50 times:

```
N, N, capital N,
```

[47] Ancient Hebrew from thousands of years ago had no spaces between words and no written vowels making it a challenging language for modern people to translate. Good spacing can make it easier to read your code.

```
we love you so much
we'll say it again.
```

Really?

String Indexes

Strings are a group of numbered characters that behave like a list. The numbering is called a character's **index**, the same word that is used for elements in a list. Indexes start with zero, just as they do in lists, and characters in a string can be referenced using indexes like elements in a list can be.

```
1                      2
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 67 8 9 0 1 2 3 4 5 6 7 8
s = " F e a r   h a s   a  l o n g  s h a d o w .  O m y ."
```

In the string numbered above, s[0] equals "F", s[22] is a period, and s[19] is "a". Try this program that uses indexed characters to make

```
s = "Fear has a long shadow. O my."
s2= s[0]+s[3]+s[1]*2+s[19]+s[12]+s[26]
print (s)
print (s2)
```

Try this program that uses loops with string indexes.

```
s = "Fear has a long shadow. O my."
for i in range(27,-1,-1):
    print(s[i], end="")
```

Palindrome Challenge:

A palindrome is a word that is the same spelled backwards as it is forwards. Write a program that asks the user for a word and checks to see if it is a palindrome. Here are some palindromes in English: **RADAR ROTATOR DEIFIED KAYAK RACECAR MADAM HANNAH** Here is one in Finnish: **SAIPPUAKIVIKAUPPIAS**(19 letters), which is Finnish for a dealer in lye (caustic soda).

Index Splicing

Python string indexes can specify a range of characters as well as just a single character. Here is how it works:

```
SubString = String[beginning character index,how many characters to take]
```

Try this program.

```
s = "Love must be sincere."
print(s)
print (s[0:3])#Prints Lov
print (s[5:9])#       must
print (s[13:20])#       sincere

print(s[:10]) #       Love must    - 0 assumed
print(s[10:]) #       be sincere.  - assumes to end
```

```
                          1                   2
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7
s = " L o v e   m u s t   b e   s i n c e r e . "
```

String functions

Here is a table of some of Python's most popular functions for strings.

Function	Purpose
Chr(integer)	Returns the **character** for the Unicode number
Ord(character)	Returns the **Unicode number** of the character
Len(string)	Returns the **length** of the string
Str(number)	Converts a **number into a string**
Int(string)	Converts a **string into a number** there must be no decimal and no alphabetical characters
Float(string)	Converts a **string into a number** with decimals. There must be no alphabetical characters

Here is a program that encodes spy messages by adding one to the Unicode number of letters.

```
from graphics import *
win=GraphWin("Cypher",400,400)

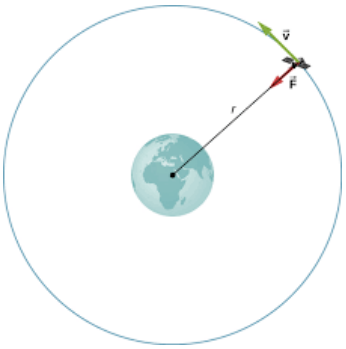
while 1:
k=win.getKey()# Wait for a key press (no print)
if len(k)>1:# handle Zelle's special characters
        if k=="space" :print(" ", end="")
        if k=="period":print(".", end="")
        if k=="comma":print(",",end="")
        if k=="Return":print("\n",end="")# \n for ENTER
    else:
        print(chr(ord(k)+1), end="")#the character after k
```

Decoder Challenge:

Using this program, write a program that takes the coded text and changes it back to readable text.

2-D Physics

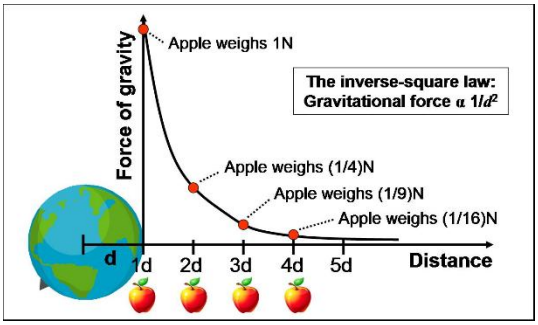
Single Point Gravitation



Gravity is a strange thing. It has puzzled scientists for years. If you are like most people, you think of gravity as a downward motion. For most situations this one-dimensional view of gravity works. In our previous discussion of gravity, the Earth's gravity **g** was added to **dy**, the vertical movement of an object. The strength of gravity never changed, in our programs. That is not true for objects far away in space. Secondly, if you think about it, because the Earth is round, there is no such thing as down. Strangely, objects that orbit the Earth are continually fallingtowards the Earth's center of gravity, but never getting any closer to the Earth.[48]

Gravity - Strength

According to Newton, gravity diminishes by the square of the number of radii an object is from the gravity source. That means a person who weighs 180 pounds on Earth's surface, (the Earth is 8,000 miles across, or 4,000 miles in radius) would weigh a quarter as much,or $1/2^2$, which is 45 pounds. At 8000 miles away,the same person would weigh only 20 pounds, or one ninth, or $1/3^2$). See the picture to the right.



To calculate the strength of Earth's gravity for an object in outer space, You will need to divide Earth's surface gravity[49] by the square of the floating object's distance from the Earth's center.Calculate the distance the object is from Earth using the Pythagorean Theorem, using the method we discussed earlier. Click here for a full discussion. Here is some Python code for figuring out the Earth's gravitational pull on an object at (x2,y2):

---

[48] Einstein's Special Theory of Relativity maintains that gravity is a distortion in time and space. Called time dialation, time actually goes more slowly of objects when they are in a gravitational field, but the difference is very small. See https://www.insidescience.org/news/time-moves-faster-upstairs.

[49] Each planet has its own gravity constant. Earths gravity constant or surface gravitational strength is 9.8 meters per second acceleration every second or 9.807 m/s^2. The Moon's is $1.62m/s^2$, or 0.16 times that of the Earth. Mars' is 3.711 m/s^2, or 0.38 times that of the Earth. Jupiter's is m/s^2, or 2.58 times that of the Earth.

```
From math import sqrt
Dist = sqrt((xEarth - x2)**2+(yEarth - y2)**2)
SpaceGravity = EarthGravity/Dist**2
```

Gravity – Direction

As we discussed earlier, gravity does not pull objects down, it pulls objects towards its own center of gravity. To send the object on its way toward the Earth's center, you will need to calculate both the horizontal and vertical pull of gravity, **gx** and **gy** in our program. This can be done using simple ratios.

Firstsubtract the Earth's x value from the object's x value to find out how far the objects are away from each other horizontally. Then divide the horizontal distance by the total distance that you got earlier using the Pythagorean Theorem. That will give you how much of the Earth's gravity is a horizontal pull,**gx**. Do the same for y to get the vertical pull of gravity,**gy**.

Here is some ratio times gravity code that will do the job.

```
gx=(x1-x2)/dist*SpaceGravity
gy=(y1-y2)/dist*SpaceGravity
```

Finally, add **gx** to **dx**, and add **gy** to **dy**. Here is a program that has a orange circle orbiting a blue circle named Earth.

```
#Single point gravity - Gravity2.py
from graphics import *
from math import sin, cos, pi, sqrt

win = GraphWin("Single Point Gravity",600,600)


cx=win.width/2
cy=win.height/2

er=20#Earth Radius
g=60  #Earth gravity at its surface

x=cx#----Object directly above the Earth
y=cy*.3# 30% of Earth's distance from top
r=10# Radius of orbiting object
dx=.3  # object starting speed across x
```

```
dy=0# object starting speed across y

s=Circle(Point(x,y),r)# -- Create the object
s.setFill("orange")
s.draw(win)

Earth=Circle(Point(cx,cy),er)#Create the Earth
Earth.setFill("dark blue")
Earth.draw(win)
#Earth=Image(Point(cx,cy),"earth3.gif")←or put a picture here
while 1:
    s.move(dx,dy)
    x=x+dx
    y=y+dy

    for d in range(90000):pass

dist=sqrt((cx-x)**2+(cy-y)**2)#-- Gravity section --
grav=g/dist**2
gx=(x-cx)/dist*grav
gy=(y-cy)/dist*grav
dx=dx-gx
dy=dy-gy
```

Circular Orbit Challenge

Our friends at SpaceX like their spaceships to travel in circular orbits. Dipping into the atmosphere and burning up is considered bad form by them. In the example above, the object starts directly above the Earth so it has only horizontal velocity. If you change the horizontal velocity to the square root of **Earth's gravity at the object's location**(not Earth's surface) times **how far the object is from the Earth**, the orbit will become a perfect circle. Here is a math formula$\boldsymbol{v} = \sqrt{\boldsymbol{Gr}}$.

Double Point Gravitation

All matter has a gravitational pull.[50] When you jump, the Earth pulls on you with its gravity moving you toward its center, but you also pull on the Earth moving it towards your own center of gravity. The amount the Earth moves is an infinitesimally small amount because it contains a great deal of matter and you are very small in comparison.

[50] The Higgs Boson, by theory is the sub atomic particle, or part of the atom, that gives an atom a gravitational pull.

By applying the Earth's gravity to the floating object, and the object's gravity to the earth, we can simulate the motion of two objects affected by each other's gravity in outer space.[51] Here is an example.

```
#Double point gravity - Gravity3.py
from graphics import *
from math import sin, cos, pi, sqrt
win = GraphWin("Single Point Gravity",600,600)

ex=win.width/2#---- Earth
ey=win.height/2
edx=-.0745
edy=0

er=20
eg=er**2/10  #Gravity based on size

ox=ex#---- objectdirectly above the Earth
oy=ey*.3
odx=.3# object initial speed (all horizontal)
ody=0

r=10
og=r**2/10

s=Circle(Point(ox,oy),r)# Create object
s.setFill("orange")
s.draw(win)

Earth=Circle(Point(ex,ey),er)#Create the Earth
Earth.setFill("dark blue")
Earth.draw(win)
while 1:
    s.move(odx,ody)# Move object
    ox=ox+odx
    oy=oy+ody

    Earth.move(edx,edy)# Move Earth
    ex=ex+edx
    ey=ey+edy

    for d in range(50000):pass
```

[51] Physics tells us that the motion of two objects is the sum of both gravities divided by the mass of each object individually. In this case, we increased the gravity to represent the difficulty of moving a bigger mass to make the program more like the single point program.

```
    dist=sqrt((ex-ox)**2+(ey-oy)**2)

#----Earth pulling on object
    egrav=eg/dist**2
    egx=(ox-ex)/dist*egrav
    egy=(oy-ey)/dist*egrav
    odx=odx-egx
    ody=ody-egy

#----Object pulling on Earth
    ograv=og/dist**2
    ogx=(ex-ox)/dist*ograv
    ogy=(ey-oy)/dist*ograv
    edx=edx-ogx
    edy=edy-ogy
```

Rotational movement

To move an object in the direction it is facing you need two things: the angle and the power of the movement. Here is some code to give you the idea.

```
dx = power * cos(a)
dy = power * sin(a)
```

Here is a program that simulates an object in free floating space with something like a rocket to propel it.

```
# Free floating rocket - Rocket1.py

from graphics import *
win=GraphWin("Rocket",1024,768)
from math import sin, cos, pi

rsize=40
x=win.width/2#Start rocket in the center
y=win.height/2
dx=0
dy=0
a=0
da=0

delay=400000
power=.3  #Power of the rocket motor

def DrawRocket(x,y,a):# <- Replace with any cool rocket drawing
    global r1, rc# Make objects global for undraw function
r1=Line(Point(x,y), Point(x+cos(a)*rsize,y+sin(a)*rsize))
```

```
    rc=Circle(Point(x,y),rsize/4)
    rc.setFill("green")
    r1.setWidth(3)
    r1.draw(win)
    rc.draw(win)

def UndrawRocket():
    global r1,rc
    r1.undraw()
    rc.undraw()

while 1:
    DrawRocket(x,y,a)

    for d in range(delay):pass

    UndrawRocket()

    k=win.checkKey()#--- keyboard commands ---
    if k=="4":#Turn counter clockwise
        da=-.02
        if a<0:a=a+pi*2
    if k=="6":#Turn clockwise
        da=.02
        if a<pi*2:a=a-pi*2
    if k=="5":#Stop turning
        da=0

    if k=="8":# Fire rocket motor to go forward
dx=dx+cos(a)*power
dy=dy+sin(a)*power
    x=x+dx
    y=y+dy
    a=a+da

    if x>win.width:x=x-win.width
    if x<0:x=x+win.width
    if y>win.height:y=y-win.height
    if y<0 :y=y+win.height
```

Rocket in Gravity Field

Here is a program that Combines the single point gravitation program, from earlier, with the rocket programmed above. Try it out.

```
# Rocket in gravity field - Rocket2.py
```


```
from graphics import *
win=GraphWin("Rocket in Gravity Field",1024,768)
from math import sin, cos, pi, sqrt

rsize=20#Rocket info---
x=win.width/2
y=win.height/4
dx=0
dy=0
a=0
da=0
power=.3# Rocket motor power

delay=400000

ex=win.width/2 #Earth Info
ey=win.height/2
er=10
eg=100

def DrawRocket(x,y,a):
    global r1, rc
r1=Line(Point(x,y),  Point(x+cos(a)*rsize,y+sin(a)*rsize))
    rc=Circle(Point(x,y),rsize/4)
    rc.setFill("green")
    r1.setWidth(3)
    r1.draw(win)
    rc.draw(win)

def UndrawRocket():
    global r1,rc
    r1.undraw()
    rc.undraw()

Earth=Circle(Point(ex,ey),er)
Earth.setFill("dark blue")
Earth.draw(win)
while 1:
    DrawRocket(x,y,a)

    for d in range(delay):pass

    UndrawRocket()

# ---- Real Time Movement -----
    k=win.checkKey()
```

```
    if k=="4":
        da=-.02
        if a<0:a=a+pi*2
    if k=="6":
        da=.02
        if a<pi*2:a=a-pi*2
    if k=="5":
        da=0

    if k=="8":
        dx=dx+cos(a)*power
        dy=dy+sin(a)*power

#---- Gravity  -----
dist=sqrt((ex-x)**2+(ey-y)**2)
grav=eg/dist**2
gx=(x-ex)/dist*grav
gy=(y-ey)/dist*grav
dx=dx-gx
dy=dy-gy

    x=x+dx
    y=y+dy
    a=a+da

#wrap on edge
    if x>win.width:x=x-win.width
    if x<0:x=x+win.width
    if y>win.height:y=y-win.height
    if y<0 :y=y+win.height
```

Using pygame to make drawings

Pygame is used by professionals to write games. It offers more options, faster execution, and more freedom than Zelle graphics, but it must be installed using pip, and it is more complicated to program.

Here is a table to convert progams from Zelle graphics to pygame.

Zelle Graphics Structure	Pygame Structure[52]

Startup
`from graphics import *`
`win = GraphWin("title", 640, 480)`
	`from pygame import *`
	`init()`
	`win = display.set_mode((640,480))`
	`display.set_caption("title")`

`win.setBackground("gray")`	`win.fill((125,125,125))# red,green,blue 0-255`

Lines
`ln=Line(Point(x1,y1),Point(x2,y2)`
`ln.draw(win)`
	`#(window,color(9,9,9),Start[9,9],End[9,9],Thickness)`
	
	`draw.line(win,(r,g,b),[x1,y1],[x2,y2],1)`
	`display.flip()`

Circles
`c=Circle(Point(x,y),r)`
`c.draw(win)`
	`#(window,color(9,9,9),[Center 9,9,width,height], Thickness)`
	
	`draw.circle(win, (r,g,b), [x,y],r,thickness)`
	`display.flip()`

Rectangles
`r=Rectangle(Point(x1,y1),`
`Point(x2,y2))`
`r.draw(win)`
	`draw.rect(win,(r,g,b),(x,y,width,height),thickness)`
	`display.flip()`

Ovals
`o= Oval(Point(x1,y1),`
`Point(x2,y2))`
`o.draw(win)`
	`draw.ellipse(win,(r,g,b),[x1,y1,r,r],thickness)`
	`display.flip()`

~~**Arcs**~~	`pygame.draw.arc(screen,color,(x,y,width,height,` `start, stop, thickness)`

Texts
`t = Text(Point(x,y), "Hello")`
`t.draw(win)`
	`fontOb = font.SysFont("Lucida Calligraphy"[53], 24)`
	`#  text, antialiasing,forground,background`
	`t=fontOb.render('Hello',True[54],(r,g,b),(r,g,b))[55]`
	`win.blit(t,(x,y))# text & location`
	`display.flip()`

Here is a sample program, so you can see how these commands work.

```
from pygame import *
init()

win = display.set_mode((400,400))
display.set_caption("Graphic Commands")

win.fill((240,240,255))  #clear to light blue

#     (window, color(9,9,9), Start[9,9], End[9,9], Thickness)
draw.line(win, (0,0,160), [100,100], [200,100], 1)
draw.line(win, (0,0,160), [100,100], [100,200], 1)
draw.line(win, (0,0,160), [200,200], [100,200], 1)
draw.line(win, (0,0,160), [200,200], [200,100], 1)
```

[52] See the Dr. P. Conrad of the University of Southern California Santa Barbara at https://www.cs.ucsb.edu/~pconrad/cs5nm/topics/pygame/drawing/ for a **good concise summary** of pygame graphics commands.

[53] The font name can be replaced by **None**. If it is, Python will use the computer's default font.

[54] True here turns on **antialiasing**. When true, the computer will smooth out edges on fonts called **jaggies**.

[55] Rgb values can be replaced by **None**, an invisible color.

```
#        window, color rgb, [x,y],    r, Thickness
draw.circle(win, (160,0,0), [150,150],50,3)

#      window, color rgb, [x,y], width, Height, Thickness
draw.rect(win, (0,200,0), (205,205,80,30),5)

#                          Font / None,       ,Size
basicfont = font.SysFont("Lucida Calligraphy", 24)
#                    Text,   Antialiase?, fore, back
t = basicfont.render('Hello World!',True,(255,255,0),(0,90,90))
#      Text, (x,y)
win.blit(t, (100,50))
display.flip()  # Put it all on the screen
```

First you need to tell python to get pygame from its library and to start it up. This is how you do that.

```
from pygame import *
from sys import *
init()
```

Now tell Python to make a new window for graphics. Do that with this command:

```
win = display.set_mode((640,480))
```

This makes a graphic window that is 640 across and 480 down. You can change the window size to anything you want by changing the numbers in the screen.

Pygame is set up differently than turtle graphics. In turtle graphics, the turtle starts in the center of the window, and the center of the window is location 0,0. Pygame does not have a turtle, and position 0,0 is the upper left of the window.

A second difference is that nothing is drawn until you give a command

```
display.update()
```[56]

Pygame does this so it can do groups of commands together, so it can run faster and prevents flickering when there is animation. Programmers call this double buffering.

[56] The difference between pygame.display.flip and pygame.display.update is, that display.flip() will update the contents of the entire display display.update() allows to update a portion of the screen, instead of the entire area of the screen. Passing no arguments, updates the entire display. https://stackoverflow.com/questions/29314987/difference-between-pygame-display-update-and-pygame-display-flip

Here are some commands to play around with:

```
win.fill(color)
```

Color is a set of three numbers in parentheses that looks like this: (120,60,180). The first number is the amount of red 0-255, the second number is the amount of green 0-255, and the third number is the amount of blue 0-255. All of the following commands would work in a program:

```
win.fill((255,0,0))   #Is a bright red screen
win.fill((0,0,40))   #Makes the screen a dark midnight blue
win.fill((250,250,0))#Makes a bright yellow out of red and green
win.fill((255,255,25))#All the colors together make bright white
win.fill((125,125,125))#We call dark white gray
```

You can even use variables to store colors by name like this:

```
orange=(255,127,0)     #Variables can be tupels[57]
win.fill(orange)
```

Hold your arm up to the screen and try to mix your skin color by changing this program.

```
from pygame import *
from sys import *
init()

win = display.set_mode((640,480))

win.fill((125,125,125))
display.update()
```

Lines are easy to draw in pygame. They are simply two points connected. Here is how the command works:

```
draw.line(window,color,1,[point1, point2],thickness)
```

Remember window is the name of the variable you used in the display command earlier.

Remember color is a tuple made of red, green and blue.

[57] **Tuples** are groups of numbers or strings. Python programmers call tuples **immutable**. That means the data they contain cannot be changed. Interestingly, the data in tuples can be deleted and new items can be added.

Points are a tuple with two numbers. The first is how far the point is from the left of the screen, the second is how far the point is from the top. So, a point that is 100 over and 50 down would look like (100,50) in Python.

Here are some line examples:

========== Under Construction ==========

Additional Ideas

Text Color

```
# Text color program
# (does not work in Thonny)

import sys

ecode=chr(27)
print('\x1b[6;30;42m' + 'Success!' + '\x1b[0m')
print (ecode+"[6;30;42m")
print ("\027\133"+"31")
print ("escseq")

print("\033[31;1;4m Hello \033[0m")
```

Random Numbers

```
#Random Circles

from random import *
import time

import turtle
win=turtle.Screen()
t=turtle.Turtle()

def eye():
```

```
    s=int(random()*85+15)
    r=s
    c=0
    for i in range(int(s/5)):

#---------random color-----------
        c=c+1
        if c>2:
            c=0

        if c==0:
            t.color('red')
        if c== 1:
            t.color('blue')
        if c==2:
            t.color('yellow')

        t.begin_fill()
        t.circle(r)
        t.end_fill()
        r=r-5
 #==========Main====================

print (int(random()*400)+1)

t.speed(0)

while 1:
    t.penup()
    t.setposition(int(random()*400-200),int(random()*400-
200))
    t.pendown()

    c=int(random()*3)
    if c==0:
        t.color('red')
    if c== 1:
        t.color('blue')
    if c==2:
        t.color('yellow')

    eye()
```

Making sounds with Beep

```
# Beep Program
# Modify to use in MAC

import winsound #  for beep - Windows API
import time

winsound.Beep(2500, 100) #Frequency in herz[58], duration in ms
time.sleep(0.3)

#This file must be loaded to where you save programs
#The number at the end 1=Async, 8=loop
winsound.PlaySound("beep02.wav",0)
time.sleep(0.3)

winsound.PlaySound("beep03.wav",0)
time.sleep(0.3)

winsound.PlaySound("beep04.wav",0)

#for MAC:
#import os
#os.system("afplay sound.wav&")
#runs a program named aplay, the & makes it play in the background
```

Here is a function that makes your Python program make sound in either MAC or windows by creating a function named **tone** that works like winsound.Beep. To use it just type it in, after you get it to work, remove the testing code and save it as "tone". Any time you want to make sounds using tone, put an **import tone** command at the top of your program.

```
#Tone Program

#This program creates a function tone(frequency,duration)
#for both MAC OS and Windows

try:
    import winsound
except ImportError:
    import os
    def tone(frequency,duration):
#apt-get install beep
```

[58] Herz, abbreviated Hz, is a measure of how many waves occur per second. Sound is actually made up of waves in the air around us. Humans can hear tones between 20 and 20,000 Hz, at best. As we age, we lose our ability to hear low and high tones.

```
        os.system('beep -f %s -l %s' %
(frequency,duration))
else:
    def tone(frequency,duration):
        winsound.Beep(frequency,duration)

#Testing code--------------------------------------------
for f in range (100,1000,50):
    tone(f,250)
```

Making sounds when things hit

Here is a fun program that combines what you learned about moving objects with your knowledge of sounds.

```
# Bounce program
# Uses Zelle Graphics
# Works in Windows using an API[59]

from graphics import *
from math import *
import winsound #  for beep

winsound.Beep(2500, 100) #Frequency in herz, duration in ms

win = GraphWin("Output", 800, 600,1)
x=100
y=100
dx=2
dy=0
d2=0
g=.003

c = Circle(Point(x,y), 10)
c.setOutline('black')
```

[59] **API**stands for **A**pplication **P**rogrammers **I**nterface. **Application programs**, sometimes called **apps**, are programs that depend upon an operating system like Windows to run. They include programs like Microsoft Office, Google Chrome, Zork 2000, and the programs you have been writing. Application programs help the user do things like write papers, do their taxes, and waste time playing games. On the other hand, operating systems like Microsoft Windows are made up of a confederation of smaller programs called **systems programs**. These programs help the computer to operate. They do things like put images on the screen, locate and load files, and bring information in from the internet. Companies like Apple, Microsoft, and IBM created the systems programs that make up their operating system with APIs to provide a way for application programs to run parts of the operating systems by calling these systems programs as functions. The one used in this program makes tones. Click here to see one that controls voltages coming out of a parallel port.

```
c.setFill('red')
c.draw(win)
print ("test")
while 1:
    dy=dy+g  #add gravity
    x=x+dx
    y=y+dy
    if x>800 or x<1:
       dx=-dx
       winsound.Beep(200,60)
    if y>600 or y<1:
       dy=-dy
       winsound.Beep(200,60)

    c.move(dx,dy)
    for d in range(60000):
        d2=d2
#time.sleep(.0001)
win.getMouse() # pause for click in window

win.close()

#main()
```

Playing music in Python

Game programs and puzzle programs frequently play music in the background. Here is how to do

```
#Music Program

import subprocess  #You need two imports
import os

#=======Here we tell Python where the program is and
#       where the music is
#Use \\ for folders - this is for windows only
path_to_prog = 'C:\\Program Files (x86)\\Windows Media Player\\wmplayer.exe'
path_to_file = 'h:\\code\\bloodytears.mp3'

#=======Here is how to call the music
subprocess.Popen([path_to_prog, path_to_file])
for i in range(300000):# <--- counts while music plays in background
    print(i)

#==== Here is how this works:
```

```
#      subprocess.call([path_to_notepad, path_to_file])
#      Run outside program:    ([program,      arguments])
```

Events and Real-Time

Real time programs run continually without waiting for the person running the program. This is the type of program commonly used by games. That way enemies can shoot at you while you wonder what to do. Python uses something called events to accomplish this.

Programmers call programs written this way event driven. This is how an event driven program works. The main logic of the program runs without interruption until the person playing the game or running the program presses a key or does something with the mouse. The computer detects it and jumps to the part of the program designed to deal with the key or mouse action. When it finishes dealing with the keypress or mouse action, it goes back to what it was doing before.

```
#Keys1
#- real time interactive program 1

import turtle
turtle.setup(400,500)
wn = turtle.Screen()
t = turtle.Turtle()
t.shape("circle")
t.color("dark blue")

x=0.0
y=0.0
dx=0.0
dy=0.0
m=0.2
t.speed(0)

def f6():
  global dx,m
  dx=dx+0.1

def f2():
   global dy,m
   dy=dy-m
```

```
def f4():
   global dx,m
   dx=dx-m

def f8():
    global dy,m
    dy=dy+m

def f5():
    global dx,dy
    dx=0
    dy=0

def fesc():
   wn.bye()

t.penup()
wn.onkey(f8, "8")    #"Up"
wn.onkey(f4, "4")    #"Left"
wn.onkey(f6, "6")    #"Right"
wn.onkey(f2, "2")    #"Down"
wn.onkey(f5, "5")
wn.onkey(fesc, "Escape")
#wn.ontimer(updt(x,y,dx,dy), 2)
wn.listen()
while 1:
    x=x+dx
    y=y+dy
    if x> 200: x=x-400
    if x<-200: x=x+400
    y=y+dy
    if y> 200: y=y-400
    if y<-200: y=y+400

    t.setposition(x,y)

wn.mainloop()



#Keys2
#- Real time interactive program 2

import turtle
turtle.setup(500, 500)
win = turtle.Screen()
t = turtle.Turtle()
```

```
sho=1

def k8():
   t.forward(45)

def k4():
   t.left(45)

def k6():
   t.right(45)

def k2():
   t.back(45)

def k5():
    if t.isdown():
        t.penup()
    else:
        t.pendown()

win.onkey(k8, "8")   #"up")
win.onkey(k4, "4")   #"Left")
win.onkey(k6, "6")   #"Right")
win.onkey(k2, "2")   #"down")
win.onkey(k5, "5")
#win.ontimer(k2,1000)

win.listen()
```

Color Mix Function

The rgb() function does not work in Thonny (a Python editor). This function will change the color using an HTML (HyperText Markup Language) command that both Thonny and Python will accept.

```
#RGB Function for Thonny

def rgb(r,g,b): #Red Green Blue -> HTML Hex Code
    r2=hex(r)[2:]
    if len(r2)<2:  r2=r2+"0"
    g2=hex(g)[2:]
    if len(g2)<2:  g2=g2+"0"
    b2=hex(b)[2:]
    if len(b2)<2:  b2=b2+"0"
    rgb2=r2+g2+b2
```

```
    return "#"+rgb2
```

Writing words in turtle graphics

Drawing words in turtle graphics is tough. Python has a write command that makes it easy. The wirte() function will let you make words of any size, color or font. Here is a program that will show you how it is done.

```
#Turtle Write

from turtle import *
win=Screen()

t=Turtle()

t.write("testing 1,2,3 testing 1,2,3 testing 1,2,3 ",
move="true")
right(90)
forward(120)
t.write("AFTER MOVE=TRUE")
t.setposition(40,-40)
t.write("20 point",font=("Arial",20,"normal"))
t.penup()
t.setposition(-100,50)
t.color("dark blue")
t.write("After color and penup")
```

Making a Pyramid

Loops and functions can produce some amazing effects. Here is a program that uses a function that will make a square any size and a second function that makes any size pyramid out of squares. Try it out.

```
#Pyramid

import turtle
win=turtle.Screen()
t=turtle.Turtle()


def sq(s): #make a square
    for i in range(4):
```

```
        t.forward(s)
        t.left(90)

def pyr(levels):  #make inset squares
    t.speed(0)
    for i in range(levels):
        sq(i*6+6)
        t.penup()
        t.backward(3)
        t.left(90)
        t.backward(3)
        t.right(90)
        t.pendown()

pyr(20)
```

Controlling Connected Devices

Computers tune and operate your car, send your voice from your phone to your friend, and map the surface of Mars. Many old computers have a parallel port that is easy to control. GPIOs (General Purpose Input Output) devices you can buy for just a few dollars will give that ability to any computer. Here is a program that will give you a start in robotics. You will need to download a program named inpout.dll and put it where you save your Python programs.

```
# Prt2
# - program to turn on ports in a GPIO or Parallel port of a computer
#requires inpout32.dll in c:\windows\system32

import ctypes
from time import *

# ------------------------------------------------------------
# Basic I/O routines - These can be used in the Python shell.

def In(addr) :
    return ctypes.windll.inpout32.Inp32(addr)
# Read byte from io address addr

def Out(addr, byte) :
    ctypes.windll.inpout32.Out32(addr, byte) # Send byte to
#io address addr

# ------------------------------------------------------------
```

```
Out (888,1)# Numbers 1,2,4,8 … turn on devices
```

Circling motion in turtle graphics

Lots of things in our wonderful world swing, orbit or arc. Math people can plot spinning objects using sine and cosine. The sine simply gives you how much to go up or down, and the cosine gives you how much you need to go back and forth. In Python and in math, they are written like this:**sin(angle)** and **cos(angle)**.

```
# Earth Moon Program

from math import sin,cos
from turtle import *
win=Screen()
m=Turtle()
e=Turtle()

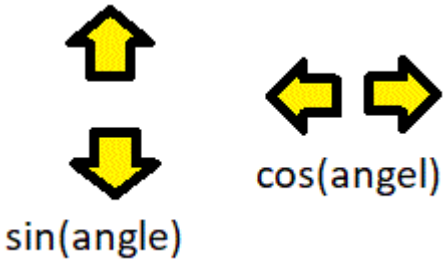
r=200.0
a=0.0
da=0.01
win.bgcolor("black")

obM="moon1.gif"   #copy the picture files to where your program is
obE="earth.gif"

win.register_shape(obM)
win.register_shape(obE)

e.shape(obE)
m.shape(obM)
e.penup()
m.penup()

while 1:
    a=a+da
    if a>3.14159*2 :a=a-3.14159*2
    m.setposition(0+cos(a)*r, 0+sin(a)*r*.25)
    for d in range(20000):pass
```



Measuring time in Python

It is possible to time how long it takes for people to answer a question, or to give them five seconds to disarm a bomb and save the whole city. (The AS command lets you rename the function to something simple to type.)

```
#timer program

from timeit import default_timer as timer

print(timer()) #<-  how long since program started

startTime=timer()
for i in range(30000):pass
print(timer()-startTime) # <- How longdid this loop take

t30=1.0/30.0 # a thirtieth of a second - eye dwell time
startTime=timer()
while (timer()-startTime)<t30: pass
print(timer()-startTime)
```

Randomness

Many philosophers believe that there is no such thing as random event or a random number. This is certainly the case with computers. They cannot help but pick the same number each time. Computer designers dealt with this by taking the time in thousandths of seconds and scrambling numbers from that first number. Try it out by using the random function below. (Notice that you can change the name of the functions by using the AS command.)

```
#Random numbers
From random import random as rnd

a=rnd()
print(a)
```

The Star Program

Here is a program that will make any kind of star, as long as it has an odd number of points. Try replacing the n=5 command with an input command and see what you can make.

```
#Star program
```

```
import turtle
win=turtle.Screen()
t=turtle.Turtle()

n=5 #<-- number of points (must be odd)

inner="yellow"
outer="red"

t.speed(0)
t.hideturtle()
n=n*2  # go twice because we are going through to the
t.penup()# other side to make the star
t.backward(200)
t.pendown()

# Change color, set to fill and do inner
t.color(inner)
t.begin_fill()
for i in range(n):
    t.forward (400)
    t.left(180-360/n)
t.end_fill()

#change color and make outline
t.color(outer)
for i in range(n):
    t.forward (400)
    t.left(180-360/n)
```

Sequential Files

Computers have two main types of memory[60]: **RAM** and disk. All programs run from RAM and up until now, it is the only memory you have used. RAM is fast, small in size, and volatile; a

[60]Modern computers have many kinds of memory. In order of how fast they are they include CPU registers, cache memory, ROM, RAM, and one or more HDDs or SSDs. The CPU registers are super-fast memory built right into the CPU which is the brains of your computer. There are 16 registers in most CPU types. Very few programmers ever deal with registers. They are hidden deep within the operating system of the computer. Cache memory (pronounced "cash") is special fast temporary memory used to speed up computer processes. Typically computers have about 6 megabytes of cache memory. Systems programmers will sometimes program the use of cache memory, but usually its use is invisible to both users and programmers. ROM or Read Only Memory, is made up of pre-programmed memory chips that help the computer know what type of hardware it has in it and how to operate its own parts. If you press a key just as your computer starts up, you can access your computer's BIOS ROM and program it to change the way your computer runs but be careful. RAM, Random Access Memory is the

word which means that it loses all its information when the computer shuts down. Disk memory is slow, huge, and nonvolatile. When you save your program, you copy the program from RAM to disk.[61]

Information saved on disk is saved in packages called files. As you probably know, the files are saved in locations named **folders or directories**. A typical computer has folders inside folders that make up a complex structure. For that reason, to retrieve or save information in a file, the computer will need a map to the location called a **path**, or it will need to look in a location it assumes you want called a **default path**.

Here is a program that writes information to the default path into a file named **file2.txt**. The period and txt are called the file's extension. Operating systems use extensions to know what type of file they are dealing with. In this case the file is a simple text file.

```
f2 = open('file2.txt','w')# Create the file to write into

for i in range(300):
f2.write(str(i+1) + " -- This is a line of text\n")

f2.close()
```

The period and txt are called the file's **extension**. Operating systems use extensions to know what type of file they are dealing with so they can open it with the right type of program. In this case the file is a simple text file. Try opening it with notepad.

Open command modes

w- Write = overwrite existing data

r- Read

a- Append = tack on to end

Adding a **+**means create it if it does not exist. For example mode a+ means add to the file, and create it if it does not exist. (All modes are lower case.)

---

memory used by programs when they are running. Finally, HDD Hard Disk Drive, or SSD Solid State Drive memory is the place information is saved to so it can be used later.

[61]Some disk memory is made out of special chips called flash memory. These drives are called solid state drives, or SSDs, for short. The chips in flash memory store enough electricity to hold on to data for several years. SSDs tend to be faster than mechanical drives and more shock resistant, but they wear out faster and eventually lose data if they remain unpowered for an extended time.

The **open command** sets up the disk so you can write to it. The first thing between the parentheses is the name of the file. The second thing listed is the mode, or what you want to do to the file. The **w**in the example above tells the computer that you want to write to the file. See the box to the right for more options.[62] In this case writing to the file also instructs the computer to erase any information that was on the file before writing to it.

After the open command there is a **for** loop that has an f2 **file object write() method** in it. When writing to a file, put the information you wish to have stored on disk between the two parentheses.

After the for loop finishes, the program closes the file using the f2 **file object close() method**. Files left open can cause strange things to happen in the computer, so good programmers usually write the open and close commands at the same time and insert code between them to make sure they don't forget to close a file.[63]

Here is a program that reads the information we wrote earlier and prints it.

```
f2 = open('file2.txt','r')#r Opens the file to read it
for rec in f2.readlines():
    print(rec, end="")
```

The **readlines()** method returns all the information fromthe entire file, in a line by line format as a list. Essentially readlines() takesthe entire file from the disk and puts it in RAM, where programmers can get to it and users can see it.

The **for** loop takes each line in the file in order. If the file is empty, the readlines() method is empty and the for loop does not execute. If you try to use readlines() a second time; it returns an empty string "". To reread the file, close it and reopen it.

Because RAM is fast, readlines() makes file processing fast, but if a file is large, the computer may run out of memory or become slow as it swaps information in and out of disk storage to make room for the big file.

Here are some Python functions that read information from disk files.

---

[62] Python has the ability to read and write **binary files** as well. Binary files are smaller and can be processed more quickly. Additionally, many types of files can only be read as binary files, but because any type of character can be written and read, programs that do binary file manipulation can damage the computer, if they do not function properly. Because end of file markers are read in as ordinary data, programmers need to be particularly careful not to read or write over other data.

[63] Many programmers prefer to use a context manager that looks something like `with open(file1.txt,r) as f1:`, and indent the file processing code. When that is done, the file closes automatically.

1. read(size)– size is an optional number that tells this functionhow many characters to read.If size if omitted, then it reads the entire file and returns it.
2. readline()–Reads a single line from file with newline[64] at the end
3. readlines()–Returns a list containing all the lines in the file
4. xreadlines()– Returns a generator to loop over every single line in the file

Here is a program that reads only one line at a time from the file. This program would be slower than the version above that reads everything into memory, but it would work for very large programs.

```
file1=open('file1.txt','r')

rec=file1.readline()#Read so we have something to check
while rec != "":# Read until no more data
     print(rec, end="")
rec=file1.readline()
```

Random Access Files

Sequential files are written from beginning to end. They must be written in order. If there is a mistake in the middle of the file, the whole file must be rewritten. Sequential files also need to be read starting from the beginning, and in order. This can be troublesome and time consuming. If the program needs the second to the last piece of data, all the data before it must be read and discarded before reading the needed data.

Random access files are like lists. Data can be read in directly from the middle of the file without reading the whole file. If a section of the file needs to be updated, it can be changed without read reading the whole file. This is made possible by the **file object seek method**. This is how it works.

```
FileObject.seek(positionInCharacters)
```

The seek method will move the position the file will read from or write to. The trick is that the position needs to be given by how many bytes it is from the beginning of the file. Bytes are enough memory to hold one character, so 100 bytes in would skip 100 letters, spaces, numbers, and punctuation marks.

One of the simplest ways to make a random access file is to make every piece of data the same size. These data chunks are called records by programmers. Here is a program that creates a

[64] New line is the Python term for character 13, the ENTER character. To type a new line character in Python either enter "/n" or concatenate chr(13).

file named "text4.txt". It writes 19 records, after that it modifies two of the records and rewrites them.

```
def field(strng, lngth):#This makes fixed length records
f=" "*(lngth-len(strng))+strng# Fill it if small
if len(f)>recLen:f=f[:lngth]# Crop it if large
    return f

def place(rec,n,lngth):#This writes random access records
file1.seek((recLen+2)*n)# +2 ->\n is two characters
    file1.write(field(rec,lngth)+"\n")

recLen=20#This is how big the records are

#Create file and erase any data -----
file1=open('file4.txt','w')
file1.close()

#Fill file ---------
file1=open ("file4.txt","w+")
for i in range(19):# Make 19 records
    place("This is record "+str(i),i, recLen)

#modify file ---------
place("5"*100,5,recLen) #put 5s in the fifth record
place("------ This is eleven 11 11",11,recLen)# change 11

file1.close()
```

Use notepad or textedit to look at the file.

Random Access Challenge

Change record 6 and 7 so they are all sixes and sevens.

This program did a good job of rewriting record 5 and 11 in the file. The next example reads as well as writes and has commented, general-purpose functions that you can modify to make your own random access file program. It includes a feature that will automatically tack records on to the end of the file, if you do not specify a record number, and an option to clear any information off, if the file needs to be a new file.

```
#This function pads strings to exact lengths
#    for random access records.  It assumes
#    a record length of 100
```

```
def field(strng, lngth=100):
    f=" "*(lngth-len(strng))+strng
    if len(f)>lngth:f=f[:lngth]
    return f

#Random Access Write supply the following
#    rfile     = file object,
#    datastr = data as a string,
#    n          = record to replace, -1 (default) = append
#    Lngth    = record size, 100 assumed
def rwrite(rfile,datastr,n=-1,lngth=100):
if n==-1:#Tack on to end assumed
rfile.seek(0, 2)# 0 counts bytes, 2 starts from
else:#        the end of the file
rfile.seek((lngth+2)*n) # \n is two characters
    rfile.write(field(datastr,lngth)+"\n")

#Random Access File Read:
#   rfile = file object
#   n = Record number to read
#   lngth = record size, 100 assumed
def rread(rfile,n,lngth=100):
    rfile.seek((lngth+2)*n) # \n is two characters
    r=rfile.read(lngth)
    return r

#Random Access File Open
#   rfName = string with file path, name, and extention
#   Erase = 0/1 - 1 erases previous information
def ropen(rfName,Erase=0):
if Erase != 0:# 1 to erase
junk=open(rfName,"w")
junk.close()
    return open(rfName,"r+")#Mode r+ will read and write


#Fill file ---------
#f1=open ("file4.txt","r+")#Mode r+ will read and write
f1=ropen("file4.txt",1)
recLen=20

for i in range(19):
    rwrite(f1,"This is record "+str(i),i, recLen)


rec=rread(f1,5,recLen)  #read example record 5
print (rec,"<--5")
```

```
#modify file ---------
rwrite(f1,"5"*100,5,recLen) #write example record 5
rwrite(f1,"------ This is eleven 11 11",11,recLen)

rec=rread(f1,5,recLen)
print (rec,"<--5")

rec=rread(f1,3,recLen)
print (rec,"<--3")

#Add records to the end
rwrite(f1,"eof------------------",lngth=recLen)
rwrite(f1,"eof2-----------------",lngth=recLen)
rwrite(f1,"eof3-----------------",lngth=recLen)

f1.close()
```

Creating a Module

A module is a Python program that you can call using an import command. It can create variables, provide code, functions and classes, and in turn call other modules. To make one, simply save code under a name and call it with import. Modules save space in the computer, simplify code, and provide specialized features. Programmers consider using modules good programming practice.

Module Challenge

Delete everything except the functions at the top of the example program above, then saveit under the name**rafiles**. (You should be able touse the randomaccess file functions in any program by using an import command.)Write a program that writes a 10 record file, prints it, then changes records 4, 5 and 6, and reprints.

Here is a program that uses the new **rafiles** module you made out of the previous example. This program creates 100 records, 100 bytes in length and prints them in reverse order.

```
from rafiles import *
f=ropen("file4.txt",1)

for i in range(100):#Create 100 records
rwrite(f, str(i)+"record -------"+(str(i)+"-")*100)

for i in range(99,-1,-1):#Print the records backwards
    print(rread(f,i))
f.close()
```

Random Access Challenge2

Change the example above so that pairs of records are swapped and it counts 2,1,4,3,6,5,8,7 and so on.

OS Commands

Operating the operating system from within your Python program.

INTERESTING AND USEFUL OS MODULE METHODS TO MANIPULATE THE OPERATING SYSTEM FROM PYTHON

	Command	Example	Description
1	listdir()	print(listdir(c:\windows))	Creates a list of all files and folders
2	path.getsize()	print(path.getsize("."))	Returns total number of bytes used by all files and folders in a directory
3	path.exists()	if path.exists("e:\")==False: print("Insert flash drive")	True if path exists, False if it does not.
4	path.abspath()	print(path.abspath(".")	Returns a string containing where the file or directory is located in the computer.
5	remove()	ifpath.exists("file.txt"): remove("file.txt") else: print("Does not exist")	Deletes a file.
6	path.isfile()	if path.isfile("food"):	True if file, False if directory
7			

```
from os import *

#Save location of current work directory
originalPath=getcwd()

#-------- cmd prompt execution ---------
system("dir")  # to run cmd commands from Python
# in this case, list the files in the dir

#-----Save output from cmd commands-----
ob=popen("dir /b").read()  #run a command and save output

#----Changing  directories (folders)----
#print (getcwd()) ← put this line in to see the change
chdir('..') # Change work directory to parent directory
#print (getcwd()) ← put this line in to see the change  (.. means the one yours is in)
chdir(originalPath) #Change dir to original as saved

# ------- User Login ----------
```


```
print(getlogin())# User login ID

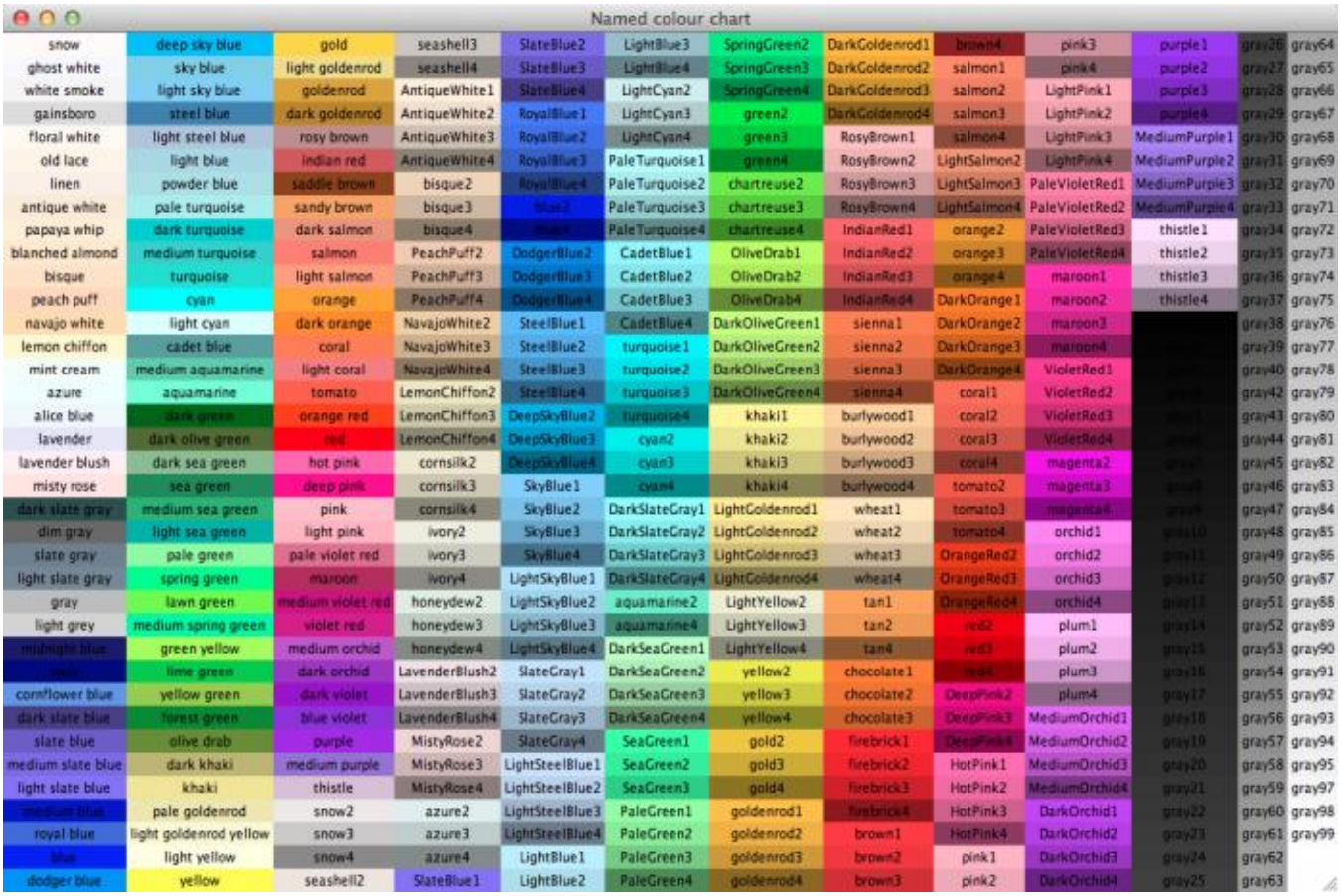
# ------ Process ID (of this program)------
print(getpid()) #process id
#-----  List all files in directory --------------
print(listdir(path=".")) #lists all files
#----  List all files in downloads directory------
print("list files in downloads----")
print
(listdir(path="c:\\users\\"+getlogin()+"\\downloads"))

#--------make new directories -----------
#makedirs(".\\testDir") #make a new folder in current dir
```

← This is the name the computer gives your program

← . means your own dir

Appendix A: Python Color Names



Appendix B

Python on-line options

These internet-based Python compilers will work on any device and at any location, but will offer slower processing and limited features compared to other options

Rating	Name	Link	Comment	Turtle	Time	Sound	Random	Sensors
	Tutorialspoint Coding Ground	https://www.tutorialspoint.com/execute_python_online.php						
	Repl,it	https://repl.it/site/languages/python_turtle		X				
	OnlineGDB beta	https://www.onlinegdb.com/online_python_interpreter						
	Python.org	https://www.python.org/shell/						
	Jdoodle	https://www.jdoodle.com/python3-programming-online						
	Rextester	https://rextester.com/l/python3_online_compiler						
	Holycross	http://mathcs.holycross.edu/~kwalsh/python/						
	Pythonanywhere	https://www.pythonanywhere.com/try-ipython/						
	Codecatemy	https://www.codecademy.com/learn/learn-python						
	Trinket	https://trinket.io/python		X	X		X	
	Skulpt	http://www.skulpt.org/						

Phone based IDEs.[65]

These are installed and resident on your phone. They are platform specific.[66]

[65] IDE stands for **I**ntegrated **D**evelopment **E**nvironment. Programmers sometimes call themselves developers. The IDE gives them a program editor and an output window all in one, making it easy to program and debug projects.
[66] Platform specific applications will only work on one kind of machine, like a PC, Mac, Iphone or android.

Rating	Name	Link	Comment	Turtle	Time	Sound	Random	Sensors
	QPython (Android)	https://www.qpython.com/	Terminal emulator , Python REPL[67] , IDE, Libraries, it runs scripts via QR code , has PIP , and courses					
	Pydroid 3 (Android)	https://play.google.com/store/apps/details?id=ru.iiec.pydroid3&hl=en_US	Simple to use, provies GUIs[68](kivy, pyQT) and will produce video games	X				
	Pythonista (Iphone)	http://omz-software.com/pythonista/	Examples, graphics, libraries, access to phone features					

[67] REPL Console: REPL stands for **R**ead **E**valuate **P**rint **L**oop; an "interpretive" program environment that executes one instruction at a time. Python normally has an REPL. If the programmer enters 2+2, Python prints 4.
[68] GUI stands for **G**raphical **U**ser **I**nterface. These allow you to click buttons using the mouse and see graphics, unlike older terminal style interfaces.

Appendix C Projects

WOW Project 1(Wonder Of Wonders)

Background:

Americans, on average, spend more than 10 hours per day in front of a screen. The entertainment industry grosses over 1.9 trillion dollars each year. There is a great deal of money to be made in the entertainment industry.

You are entering an app contest. The goal of your app is to entertain those running it. Your audience is high school age students and your instructor. Be careful, students your age can be brutal graders if there are errors, so test every aspect of your app. You, yourself will participate in the grading, but your grades themselves will be graded for accuracy.

Directions:

Members	Score Times
1	1.8
2	1
3	0.75

Design and write a program to amaze and amuse those using it. The topic is completely up to you. You can work **alone or in a group** of two or three, but groups are graded by how many members they contain. (See the table to the right.) Your app **must be written in Python**. Feel free to use Thonny, Trinket.io or any Python compiler you wish, but your program must be easily accessible by students and the instructor. **Write instructions for accessing your program** and test them with friends. Inaccessible entertainment gets no credit.

Grading:

You will receive one grade from the instructor, a second grade from contest results, and a third grade from you own evaluation of the programs entered in the contest. Students from other classes are often invited to join in evaluations of contest entries, and you may be asked to grade contest entries from other classes.

A sample grading rubric is provided on the page that follows this one. Enter grades 1 through 4, with 1 being low, and 4 being high. Try to average as close to 2½ as you can. That is the average the grading computer is set up to expect.

Scores are calculated using totals, then multiplied by the group modifier, and run through a curving algorithm. A Microsoft Excel sheet will be available to do all the math for you. It is usual to set the curve average to a B level. Sometimes hardships occur. It can be tempting to grade special cases with compassion. Grade what is there, the instructor will make modifications in special case,s so injustices do not occur.

WOW Project Grading Rubric:

Grade 1-4 enter 0 if missing

Description To Right	1	2	3	4	5	6	7	8	9	10	11	12	12	14	15	16	17	18	Subject	
1.																			Text	Beauty
2.																			Graphics	
3.																			Color	
4.																			Complexity	
																			<- TotalAdd the above	
1.																			Loops	Programming
2.																			NestedLoops	
3.																			Conditionals	
4.																			Functions	
5.																			Variables	
																			<- TotalAdd the above	
1.																			Math	Advanced
2.																			Animation	
3.																			Edge of Screen	
4.																			Control	
5.																			Timing	
																			<- TotalAdd the above	
1.																			Scope	Sum
2.																			Knowledge	
3.																			Design	
																			<- Total Add the above	
																			<- Grand Totals Add all totals	

1 Sum of Grand Totals

2 Number of student projects:

3 Average: Divide box 1 by box 2

Home AutomationProject 2

9007210: 16.05, 17.01, 17.06, 17.09, 17.11, 18.01, 18.02 20.01, 22.04 23.01

Background:

First built in the 1960s, smart buildings use a computer to control the temperature and lighting in individual rooms to save energy and to cater to the convenience of the people who use them. Smart houses now allow owners to automate lighting, AC, heating, appliances and entertainment using mobile apps from their phones.

Computers control output ports by checking the content of certain locations in memory. If your computer is old enough it might have a parallel port. The location of the parallel port is **888**. If it is a newer model, you need a GPIO (General Purpose Input-Output). These usually plug into a USB (Univeral Serial Bus) port and need to have a driver (a small program that connects the device to the operating system). Once you have the driver installed, find out what resources are used using the device manager. You can run the device manager from the control panel.
Ports will either be 5 volts DC or 0 volts DC depending upon the number you put in the memory location of the device. Use the table to right for controlling connected electronics. If you add the numbers, the computer will turn on more than one device. So, 3 would turn on both port 1 and 2, and 28 (4+8+16) would turn on ports 4, 5, and 6.

1	Port 1 on
2	Port 2 on
4	Port 3 on
8	Port 4 on
16	Port 5 on
32	Port 6 on
64	Port 7 on
128	Port 8 on

Assignment:

Step 1 – Get the DLL (Dynamic Link Library) and save it to where you keep your programs.
Using an API[69] named **inpout.dll**, write a Python program to control four lights wired to a TRIAC[70] array.
To get inpout.dll go to:
http://www.highrez.co.uk/downloads/inpout32/
Scroll down to binaries, and click the first link titled:
Binaries only - x86 & x64 DLLs and libs. (Mirror)

Step 2 – Copy the function calls into your program.
Type or copy the following import and functions to the top of your program:

```
import ctypes

def In(addr) :
    return ctypes.windll.inpout32.Inp32(addr)
```

[69] API stands for Application Programmer's Interface. Click here for a fuller explanation.
[70] TRIAC is an abbreviation for triode, because it has three wires (called terminals) coming out of it, and because it switches high voltage AC electricity on and off. It is a type of thyristor. See https://www.electronics-notes.com/articles/electronic_components/scr/what-is-a-triac.php to learn more.

```
def Out(addr, byte) :
    ctypes.windll.inpout32.Out32(addr, byte)
```

Step 3 – Calling the function.

These functions are called the same way all functions are called in Python. Simply type the name and put the right numbers or variables between the parentheses. Here are some examples:

```
Out (888,1)      <- This turns on port 1
Out (888,28)     <- This turns on port 4,8, and 16
Out (888,n)      <- This turns on port n
```

Grading and Objectives:

Points will be awarded according to the table below:

	Project	Description	-- Points--		Sgn
			First	Follower	
1	**Console control**	Turn on lights using numbers typed in from the console.	**2**	**1**	
2	**Light sequencer**	A light sequencer will turn each light on and off in order. When the last light is turned on and off, it starts over with the first light. It runs the lights in continual loop until stopped.	**6**	**3**	
3	**Button control**	Labeled graphic buttons control each light. When clicked, the light will turn on if it was off and off if it was on.	**9**	**5**	
4	**Timed light**	Turn one light on and off at specific times. The times should be programmable	**9**	**3**	
5	**Radio button control**	A modification to button control that turns off all lights except the one you click to save energy.	**6**	**2**	
6	**Dancing Lights**	Play a tune and make the lights come on depending upon the note played. Demonstrate your program with “Mary had a little lamb”.	**12**	**4**	
7	**Your device**	Bring in a device from home and run it from your computer. (Only one device per person.)	**2**	**-**	
8	**Appliance Organ**	Program the keys of the keyboard to run small devices (<50 Watts) plugged in to run, or play their note. The devices should only play while the key is pressed and should stop when the key is released.	**18**	**8**	

9	**Your own**	Student defined. (Get your idea approved.)	**0-20**	**1/3**	

You need 12 points for an **A**, 9 points for a **B**, 6 points for a **C**. You can go above an **A**, as you do with tests. Student groups split credit. Learn from anyone willing to teach.

Turn this sheet in with signatures for credit.

BiomechanicsProject 3

9007210: 16.04,13, 20.02-04, 21.02-04, 22.01,04, 23.01-04

We, as human beings walk on two legs. Biologists would tell you that from an animal's viewpoint, humans are naturally acrobatic. Programmers, however, view human movement as simpler than animal movement, as we have already seen in our work up to this point. In this project, we are going to make a computer video game that uses stick men. Feel free to use the internet; feel free to use Explore, the class text, but always respect copyrights.

Define your project

(Grade1, by Inspection)

1. Choose a partner **you have never worked with before.** (Teams of two only, please.)
2. Decide on a game or video presentation topic with your partner.
3. Submit the name of your game or video presentation along with how to win or operate it to your instructor for a grade.

Design your Project

In this project, the design is yours. The process is yours. The subject matter is yours. We only ask that you keep your work suitable for any audience and that you not defame anybody or any institution.

Your goal is to amuse. Your audience is the instructor, your parents and, eventually, the general public.

Eventually you and your partner will make this program into a phone app, so consider phone screen sizes as you design your program. Your program needs to be attractive and usable on both a computer and a phone.

Brainstorm together about how to break down the work into parts that can be written as functions.

- You will need to **assemble the project** from the sections of code and function that you both have produced. Inform the instructor how you will accomplish assembly of the program, as it will affect grades.
- Feel free to **draw** out how the program will work. Show this to the instructor, as well.
- Work out all the **math** and get help where you need it. The instructor would be happy to help you with math ideas and principles. Be sure the instructor knows who solved mathematical issues, for grading.
- Be sure **your name** is on all the work you do by using # comments. In the main program **use comments** to list your names and who assembled the program from the functions you each wrote.

Suggestion1: Send each other functions and program fragments using email attachments, a flash drive or the cloud.

Suggestion2: The instructor is a shared resource. Ask questions and ask for how to examples.

Code for a computer

(Grade2, e-mail your program)

Use Thonny with Zelle's graphics package. Write small portions of code and test them, then incorporate them into larger parts of the program. Test and debug everything. When you finish, email the project to your instructor.

Make it an App

(Grade3, by Inspection)

Use a phone based compiler with an IDE like **Pydroid 3** for android or **Pythonista** for iPhone.[71] Remember to include Zelle Graphics when you install to a phone.[72] Email your Python program to yourself and copy and paste it into the compiler.

Grading

You will be graded on scope, knowledge, robustness, and design.

1. **Scope** is the size and complexity of your program.
2. **Knowledge** is any, and all knowledge evident from your program.
3. **Robustness** is how much was debugged and how good is the user interface.
4. **Design** answers the question, is it fun and is it a good one of what it is.

[71] You are not limited to the examples given above.

[72] Programmers call packages and modules that need to be included for their program to run dependencies. It is vital these be installed and be available to the program at execution time. It is vitally important to consider the demands all the dependencies make on the computer. If a small dependency requires high level computing power or demands unusual video characteristics, it will limit what economists call your customer base, or the people who could buy and use your program.

Innovate Project 4Mobile App Development

9007210: 15.01-04,16.12,15, 17.01,03,05,09,11 18.01, 20.01,04, 23.01-03, 24.02-05, 25.03,05,07

Overview:

You will be assigned to a group of students. The goal is to produce an android phone-based app to help high school students in school. The students may be athletes, incoming freshmen, club members – any group you wish or all groups together.*Use the sheets that follow to guide you.*

Your app must:

1. **Produce the location of the school**
2. **Provide a school bell schedule**
3. **Connect you with**
    a. **school administration**
    b. **special programs**
    c. **school news**

Programming:

We will be using **MIT's App Inventor 2**[73]. Follow the instructions to the right to get the compiler to work. Google will give you a built-in place to save. App Inventor 2 has the ability to run as an interpreter[74] and to produce both **aia** and an **apk** files.

An **aia** file contains the source code[75]. This is the program you see on the screen. You can use this file to combine work from other students in your group with your own work. It is a requirement for your grade and the contest. Email it as an attachment to anyone who needs your on-going work.

How to Run MIT's App Inventor 2

1. **Make an account with Google, if you do not have one, and sign in.**
2. **If you have an android phone, go to play store and install MIT AI2 Companion.**
3. **On your computer, go to http://ai2.appinventor.mit.edu**
4.
5.
6.
7.
8.
9.

Special thanks to D. Lugo, IT specialist, for assistance in preparing this section.

[73]MIT App Inventor 2 is provide free of charge by MIT,software and support licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License © 2012-2013 Massachusetts Institute of Technology

[74] Remember, interpreters execute programs line by line, so they must be present while the program is running, and they will run programs slowly. Compilers translate the whole program at once to executable code. Compilers take a long time to make the translation, but once made, the code is fast and will run without the compiler present.

[75] Source code is the program you write or edit, the one you see on your screen when you are programming. Source codes have different extensions for example, C is .c; Python is .py.

An **apk** file is a compressed binary executable[76] for android phones. If you send the **apk** to anyone's phone, it will they will have your program as a full running app. They will not need App Inventor 2. This file is also a requirement for your grade and for the contest.

[76] An executable is machine code that runs the computer or device. If you look at it, it will look like gibberish. Executables on a PC end with .com, or .exe, but on an android, they end with .apk.

Student Teams:

Group1:

- Giovanni Carter *
- Rony Sanchez

Group 2:

- Hector Erazo *
- Lanyah Leon
- Jose Marrero

Group 3:

- Angel Mendoza *
- Yander Diaz
- Alexander Vega-Heredia

Group 4:

- Justin Jova *
- Kinsley Lafrance
- Demetrius Flores

Group 5:

- Imanol Ordaz-Paiz *
- Victor Aguiar
- Adrian Rolda

* Students with an asterisk after their names are group leaders. They **assign** tasks, **check** work coming in and see to it that the project is **complete**. Part of their grade is determined by how effectively the student in their teams contribute to the final project. They are the people who put the whole thing together. Team members report to them.

INNOV@TE CTE CHALLENGE
MOBILE APPLICATION DEVELOPMENT
Notes for The Video Presentation (3 min)

Directions: Fill in the information below, proof it, read it aloud and write a video script from it, then have the instructor check your work. Erase all directions when you finish. They are green.

How we designed the app:

Who we aimed the app at:

Examples: Students coming in from middle school, students new to the area, freshmen, seniors, athletes, magnet students

How the program will help them:

Visual effects:

Audio effects:

Unique form:

Unique Functions:

Coding language and why:

MIT's App Inventor 2

program abstraction, access to sensors, pre-programed objects, easy packaging and deployment, zero cost, and minimal installation requirements

Copyrights:

INNOV@TE CTE CHALLENGE
MOBILE APPLICATION DEVELOPMENT
Oral Presentation Text (5-7 min)

Directions: (All directions are in green. Be sure to erase them). Make a five to seven minute oral presentation on:

1. **WHAT THE PROGRAM DOES – WHY IT IS NEEDED.**
2. **HOW THE PROGRAM WAS DEVELOPED – WHY THAT IS THE BEST WAY.**
3. **SOFTWARE USED – WHY IT IS RESPECTED.**
4. **HOW THE PROGRAM WORKS.**
5. **COPYRIGHTS AND COPYRIGHT LAW.**

Visual aids: phone, pictures, projections.

Standard Outline:

I. **Introduction catch phrase**: "Have you ever…", "Did you know… "

II. **Initial summary paragraph**: "There are three things …", "To be successful in this life …"

III. **Body**:
   a. "XYZ meets these needs by …"
   b. "XYZ was developed using … specifically so … and therefore it never … and it always …"
   c. "XYZ was produced by … with … all rights are reserved … side effects include …"

IV. **Conclusion**: "Use XYZ for your …, …, and … and you will be surprised that [amazing thing]."

Be ready for questions like the samples below.

Write out questions and answers, and quiz each other.

1. **How did you make …?**
2. **What problems did you run into, how did you solve them?**
3. **What was your biggest challenge?**
4. **Where did you get …?**

Erase this box and replace it with a QR for the program on-line. Use https://www.the-qrcode-generator.com/

INNOV@TE CTE CHALLENGE
MOBILE APPLICATION DEVELOPMENT
File Submission

Erase this box and place one or more phone shaped pictures of the app doing its best, most visually impressive work here. Wrap text using the square option around all pictures.

Directions: Fill in the information required by Innov@te by editing this sheet. Write your answers in paragraph form. Take it to your instructor and read it aloud, edit it and fix all grammar and spelling until it sounds intelligent and it flows, then turn it into a script for the video. When you finish, erase all the directions. They are green text.

App Name:
Description:
Features:
Development:
Copyrights:

INNOV@TE CTE CHALLENGE
MOBILE APPLICATION DEVELOPMENT
Contest Check-Off List

Contest Date: Saturday, 12/1/19
Contest Location:
Miami Dade College Wolfson Campus

Prepared:

- ☐ App written, named and tested
- ☐ App bug free
- ☐ App has
  - school location
  - bell schedule
  - administration contact info
  - school special programs
  - school organizations,
  - school news
- ☐ App states license for use and terms of use
- ☐ There is a QR[77] and picture of the GUI[78] on the file submission page
- ☐ Team Registered – App name, members, school, grade
- ☐ Transportation arranged.

Packaged and ready the day before:

- ☐ Video presentation USB
- ☐ Tested laptop computer, charger, extension cord
- ☐ Tested demonstration phone
- ☐ Code in file format on USB (aia) and printed
- ☐ Stand-alone program on USB (apk)
- ☐ Written & practiced oral presentation timed to 5-7 min.
- ☐ File submission form on USB (pdf)
- ☐ File submission form printed

[77] **QR** is a square bar code developed by Japanese auto manufacuters in 1994. QR stands for **Q**uick **R**esponse. Most smart phones can read a QR and search its contents on the internet. See https://en.wikipedia.org/wiki/QR_code for more information.
[78] **GUI** stands for **G**raphical **U**ser **I**nterface, or what people see. Put in a picture of your program at its best.

BiomechanicsProject 5

9007210: 16.04,13, 20.02-04, 21.02-04, 22.01,04, 23.01-04

We, as human beings walk on two legs. Biologists would tell you that, from an animal's viewpoint, humans are naturally acrobatic. Programmers, however, view human movement as simpler than animal movement, as we have already seen in our work up to this point. In this project, we are going to make a computer video game that uses ideas you got from the stick man, bomb and explosion examples in class. Feel free to use the internet; feel free to use the Explore text, but always respect copyrights.

Define your project (Grade1, by Inspection)

4. Choose a partner **you have never worked with before.** (Teams of two only, please.)
5. Decide on a game or video presentation topic with your partner.
6. Submit the name of your game or video presentation along with how to win or operate it to your instructor for a grade.

Design your Project

In this project, the design is yours. The process is yours. The subject matter is yours. We only ask that you keep your work suitable for any audience and that you not defame anybody or any institution.

Your goal is to amuse. Your audience is the instructor, your parents and, eventually, the general public.

Eventually you and your partner will make this program into a phone app, so consider phone screen sizes as you design your program.[79] Your program needs to be attractive and usable on both a computer and a phone.

Brainstorm together about how to break down the work into parts that can be written as functions.

- You will need to **assemble the project** from the sections of code and functions that you both have produced. Inform the instructor detailing how you will accomplish assembly of the program.[80]
- Feel free to **draw** out how the program works. Show this to the instructor, as well.
- Work out all the **math** and get help where you need it. The instructor would be happy to help you with math ideas and principles. Feel free to use the internet, but be sure to keep a list of on-line sources and credit them in your program. Be sure the instructor knows who solved mathematical issues, to ensure accurate grading.

[79] Because graphics commands differ between different devices, professional programmers often put them in functions that can be modified to port their programs to new devices quickly.

[80] That makes partial credit grading possible in the event that your program never comes together.

- Be sure **your name** is on <u>all the work</u> you do by using # comments. At the top of the program **use comments** to list both of your names and indicate clearly which one of you assembled the program from the functions and code fragments you each wrote.

Suggestion1: Send each other functions and program fragments using email attachments, a flash drive or the cloud.
Suggestion2: The instructor is a shared resource. Ask questions and ask for **HOW TO** examples.

Code your project for a computer (Grade2, e-mail your program)

Use Thonny with Zelle's graphics package. Please remember to write small portions of code and test them before incorporating them into larger parts of the program. Test and debug everything. When you finish, email the project to your instructor for a grade.

Make it an App (Grade3, by Inspection)

Use a phone-based compiler with an IDE like **Pydroid 3** for android or **Pythonista** for iPhone.[81] Remember to include Zelle Graphics when you install to a phone.[82] You may use MIT's App Inventor 2, but you will need to learn how to use the canvas, and do a great deal of translation, if you choose to use their program.
Email your Python program to yourself and copy and paste it into the compiler. Get working and compile it for distribution.

Grading Rubric

You will be graded on scope, knowledge, robustness, and design.

5. **Scope** is the size and complexity of your program.
6. **Knowledge** is any, and all knowledge evident from your program.
7. **Robustness** is how much was debugged and how good is the user interface.
8. **Design** answers the question, is it fun and is it a good one of what it is.

	Category Name	Description	wt	Good	Bad	Challenge
1	Scope	size complexity thinking	3	large, complex, clever, many parts, subs, files, etc. detailed pictures, many options	repeated code, fluff, unexecuted code, short, trivial, bad or no graphics, few options, incomplete	instructor good students
2	Knowledge	code error free features	2	"best way" programming, many sensors used, cool techniques, optimized, advanced math, animation	clumsy code, easy to program style, inefficient, low performance, bad or no animation, incomplete	instructor
3	Robustitude	user interface error free exhaustive	1.5	No errors, intuitive controls, handles bad and weird cases, many issues solved for its scope	Syntax errors, "undocumented features," obvious program options or features missing, bad timing, incomplete	emotional people
4	Design	design conventional fun	1.5	Exhausts idea, interesting, easy to use, different, unique features, interesting options	disappointing, frustrating, few options, useless, unclear, ugly, incomplete	students

[81] You are not limited to the examples given above.
[82] Programmers call packages and modules that need to be included for their program to run dependencies. It is vital these be installed and be available to the program at execution time. It is vitally important to consider the demands all the dependencies make on the computer. If a small dependency requires high level computing power or demands unusual video characteristics, it will limit what economists call your customer base, or the people who could buy and use your program.

Biomechanics Grading

9007210: 16.04,13, 20.02-04, 21.02-04, 22.01,04, 23.01-04

Define your project (Grade1, by Inspection)

										← Names
										←Grade
										Team of two, both agree
										Project name chosen
										How to operate
										Clear Goal
										EC – Math, Drawing, Code

Design, Code, and Submit Project For PC or Mac (Grade2 by e-mail)

- **Assembly, drawings,math, name,comments**

										← Names initials
										←Grade
										Scope -
										Knowledge -
										Robustiude -
										Design -
										EC – Innovations
										---- Notes ----

Make it an App (Grade3, by Inspection) Pydroid 3 /Pythonista/ MIT's App Inventor 2 …

										← Names initials
										←Grade
										Scope -
										Knowledge -
										Robustiude -
										Design -
										EC – Innovations
										---- Notes ----

ClockProject 6

9007210: 16.04,13, 20.02-04, 21.02-04, 22.01,04, 23.01-04

We are obsessed with time. Our schools have timed bells; our traffic is run on timed lights; our entertainment is run in time slots. In section 1, you are going to retrieve the system time and use it to make a clock. Each level gives you points in the gradebook. In section 2, you will research management models used by software companies, and draw pictures of how they work for a grade. Then in section3, you will use what you learned to design, code, debug and deploy a clock app. There are three grades for this project.

1 MathPython Clock Program Learn the Math by Using It (Pairs)

Work **with a partner** in this section. (Groups of two only.) Divide up the work and email parts to each other. Programmers like to divide the work up into functions to make putting the project together easy. **YOUR NAME MUST BE ON ALL THE WORK YOU PRODUCE**, not your group names. Use # comments to label the sections of code you produce. No credit will be given to code, if there is not a name attached to it.**Keep this page for grading.**

Level 1: (1 pt)

Write a program to turn the entire screen into a large accurate digital clock. Your clock must be colorful and easy to read from across the room.

Here is a program that retrieves the date and the time to help you get started.

```
import time

while 1:
    t=time.localtime(time.time())
    print(t[3],":",t[4],":",t[5])

"""
  [0] -> year
  [1] -> month
  [2] -> day
  [3] -> hour
  [4] -> minute
  [5] -> second

"""
```

Level 2: (3pts)

Make a simple analog clock with hour, minute, and second hands that keep accurate time.

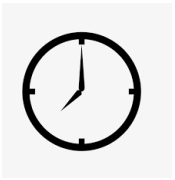
Here are some commands you may find useful:

`turtle.pensize(width)` Sets or returns the line thickness.
`turtle.setheading(to_angle)` Turns the turtle to any angle

Hint: The turtle turns counting 360° as a full turn. A clock counts 60 tics as a full turn for minutes and seconds, and 12 as a full turn for hours

Level 3: (1pt)

Put in hash marks for 12, 3, 6, and 9



Level 4: (2pts)

Change your clock so the hour hand moves small amounts as minutes elapse, so it can point between the numbers the way a real clock does. If your program works properly, the hour hand would be half way to the next hour when the minute hand reaches 30.

Level 5: (2pts)

Use math to fill in 60 small hash marks for every minute. The hash marks must always aim away from the center of the clock.

Level 6: (3pts)

Use trig to put in clock face numbers (1-12).They must be positioned correctly.

Extra Options = Extra Credit:

1. Put in the date. (1pt)
2. Put in the day of the week. (1pt)
3. Calculate the phases of the moon and display the moon. (6pts)
   or calculate the tides and display them.
4. Run an animation in the background. (2pts) with trig (5pts)
5. Your own idea (?pts)

Program Grading:

Your grade will be calculated by adding half the number of points your group gets, rounded down, plus all the points you produce for your group. (A=12, B=9, C=6) Additionally, the first team to finish[83] gets 4 extra points – two points per person.

2 Software DevelopmentModels (Pairs)

Software companies try very hard to make high quality software for low cost and high profit. During the 1980s and 1990s the industry came into crisis. Hardware was being developed at such a fast speed that the software could not keep up. Software companies made changes to the way programmers create software.

1. **Research** some of these methods. Use some of these phrases to get started:
    - Software Development Cycle
    - Waterfall
    - Agile
2. **Draw** a picture of how each works. You can use the computer, or do it free hand. (You need good pictures of at least four models for an **A**)
3. List the **advantages and disadvantages** for each model in the picture or to the side of the picture. (At least three of each per picture are needed for an **A**)
4. **Choose** a development model for the next project. Explain why you chose the model in a sentence at the bottom of the page. Put your name on your work and turn it in for a grade.

Model Grading:

Your work will be graded by the rubric:

1. Content and accuracy of pictures
2. Advantages and disadvantages are sufficient to decide between the models
3. Art quality & attractive design
4. Reasoning for decision

3 App Inventor 2Clock Program Contest Your Design (Individual)

Use the MIT App Inventor 2 to create a clock app that will work on an Android Phone. The design is completely up to you. The way you code your clock is completely up to you. When you finish, install your app to the MIT AI2 Gallery under **RidgeClock+YourNumber**. You are allowed to help each other[84], but your work must be

[83] By finished, it is assumed that at least one person has a B or better.
[84] Please remember that this is a contest. The people you help are your competitors. Kindness still dictates that you help those who ask.

your own. You may research on-line, but your work must be your own. **DO NOT COPY SOMEONE ELSE'S CODE** and put your name on it.[85] Be honest.

App Grading:

You will be graded in four areas by both your fellow students and by the instructor:

	Category Name	Description	wt	Good	Bad	Challenge
1	**Scope**	**size** **complexity** **thinking**	3	**large, complex, clever, many parts, subs, files, etc. detailed pictures, many options**	**repeated code, fluff, unexecuted code, short, trivial, bad or no graphics, few options, incomplete**	instructor good students
2	**Knowledge**	**code** **error free** **features**	2	**"best way" programming, many sensors used, cool techniques, optimized, advanced math, animation**	**clumsy code, easy to program style, inefficient, low performance, bad or no animation, incomplete**	instructor
3	**Robustitude**	**user interface** **error free** **exhaustive**	1.5	**No errors, intuitive controls, handles bad and weird cases, many issues solved for its scope**	**Syntax errors, "undocumented features," obvious program options or features missing, bad timing, incomplete**	emotional people
4	**Design**	**design** **conventional** **fun**	1.5	**Exhausts idea, interesting, easy to use, different, unique features, interesting options**	**disappointing, frustrating, few options, useless, unclear, ugly, incomplete**	students

Student grading instructions:

1	Worst in category
2	Very Bad
3	Bad
4	Low Average
5	Average
6	High Average
7	Good
8	Excellent
9	Best in category

Grade each app off the gallery with a partner who has a phone. Rate each app using a 1-9 scale, where 1 is the lowest and 9 is the highest. To make a good grade you must be careful to **average as close to 5 as possible** in each column.[86] You must give out **exactly one 1** in each column to the worst app in each category, and you must give out **exactly one 9** to the best app in each category. If you have graded correctly you will have exactly one 1 and one 9 in Scope, and one 1 and one 9 in Knowledge, and so on. When you finish evaluating the apps on paper, you will enter your evaluations into the grading computer, and the computer will grade your evaluation.

[85] Programmers usually make money by writing programs. They view taking their code as stealing, and may sue, if they can prove they lost money because of your actions.

[86] If you don't the grading program will change your numbers so they average 5, then every number you enter will have fractions added and subtracted from it and you will be inaccurate in each entry. Computers are fair graders, but ruthless.

Dirty rect animation.

https://www.pygame.org/docs/tut/newbieguide.html

The most common cause of inadequate frame rates in pygame programs results from misunderstanding the `pygame.display.update()` function. With pygame, merely drawing something to the display surface doesn't cause it to appear on the screen -- you need to call `pygame.display.update()`. There are three ways of calling this function:

- `pygame.display.update()` -- This updates the whole window (or the whole screen for fullscreen displays).
- `pygame.display.flip()` -- This does the same thing, and will also do the right thing if you're using `double-buffered` hardware acceleration, which you're not, so on to...
- `pygame.display.update(arectangleorsomelistofrectangles)` -- This updates just the rectangular areas of the screen you specify.

Most people new to graphics programming use the first option -- they update the whole screen every frame. The problem is that this is unacceptably slow for most people. Calling `update()` takes 35 milliseconds on my machine, which doesn't sound like much, until you realize that 1000 / 35 = 28 frames per second *maximum*. And that's with no game logic, no blits, no input, no AI, nothing. I'm just sitting there updating the screen, and 28 fps is my maximum framerate. Ugh.

The solution is called 'dirty rect animation'. Instead of updating the whole screen every frame, only the parts that changed since the last frame are updated. I do this by keeping track of those rectangles in a list, then calling `update(the_dirty_rectangles)` at the end of the frame. In detail for a moving sprite, I:

- Blit a piece of the background over the sprite's current location, erasing it.
- Append the sprite's current location rectangle to a list called dirty_rects.
- Move the sprite.
- Draw the sprite at it's new location.
- Append the sprite's new location to my dirty_rects list.
- Call `display.update(dirty_rects)`

The difference in speed is astonishing. Consider that SolarWolf has dozens of constantly moving sprites updating smoothly, and still has enough time left over to display a parallax starfield in the background, and update that too.

There are two cases where this technique just won't work. The first is where the whole window or screen really is being updated every frame -- think of a smooth-scrolling engine like an overhead real-time strategy game or a side-scroller. So what do you do in this case? Well, the short answer is -- don't write this kind of game in pygame. The long answer is to scroll in steps of several pixels at a time; don't try to make scrolling perfectly smooth. Your player will appreciate a game that scrolls quickly, and won't notice the background jumping along too much.

A final note -- not every game requires high framerates. A strategic wargame could easily get by on just a few updates per second -- in this case, the added complexity of dirty rect animation may not be necessary.