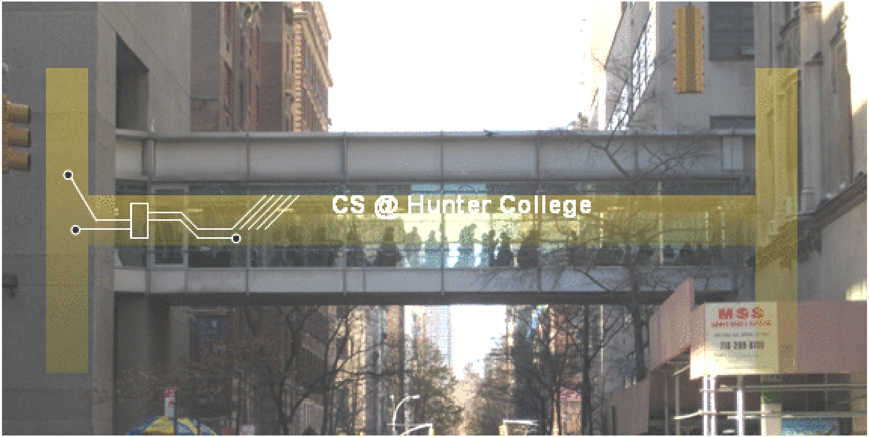
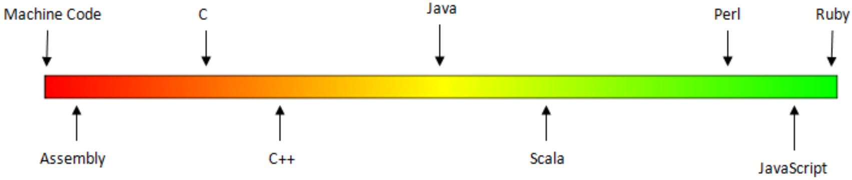
CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

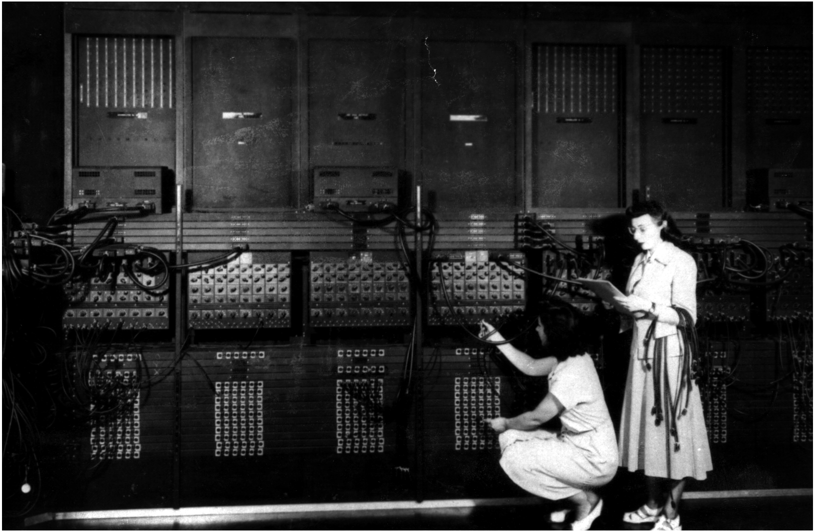
Low-Level vs. High-Level Languages



(codeCommit)

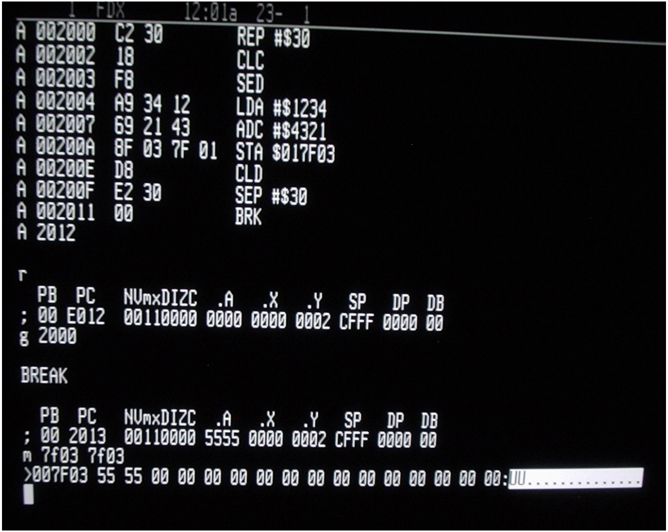
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.
- Some languages, like C, are in between– allowing both low level access and high level data structures.

Machine Language



(Ruth Gordon & Ester Gerston programming the ENIAC, UPenn)

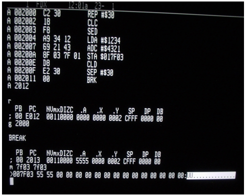
Machine Language



```
    1  FDX       12:01a  23-  1
A 002000  C2 30         REP #$30
A 002002  18            CLC
A 002003  F8            SED
A 002004  A9 34 12      LDA #$1234
A 002007  69 21 43      ADC #$4321
A 00200A  8F 03 7F 01   STA $017F03
A 00200E  D8            CLD
A 00200F  E2 30         SEP #$30
A 002011  00            BRK
A 2012

r
  PB  PC   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 E012  00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

  PB  PC   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 2013  00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00:UU..............
```

(wiki)

Machine Language

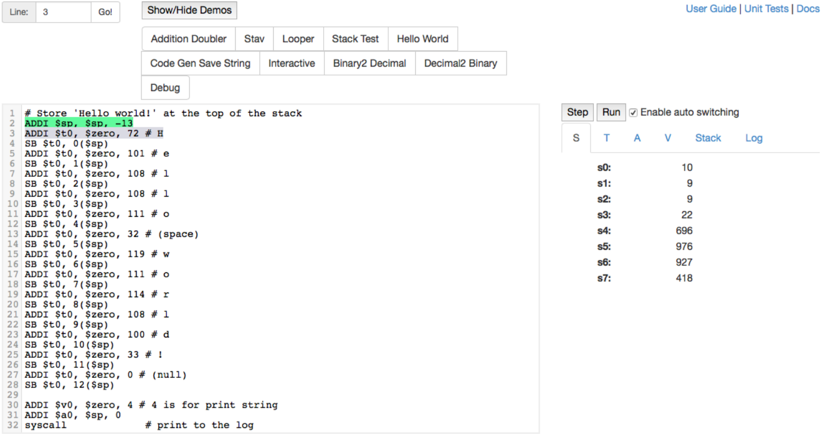


```
     1  FDX       12:01a  23-  1
A 002000  C2 30        REP #$30
A 002002  18           CLC
A 002003  F8           SED
A 002004  A9 34 12     LDA #$1234
A 002007  69 21 43     ADC #$4321
A 00200A  8F 03 7F 01  STA $017F03
A 00200E  D8           CLD
A 00200F  E2 30        SEP #$30
A 002011  00           BRK
A 2012

r
  PB  PC   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 E012  00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

  PB  PC   NVmxDIZC  .A   .X   .Y   SP   DP  DB
; 00 2013  00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00:UU..............
```

(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
- It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
- Due to its small set of commands, processors can be designed to run those commands very efficiently.
- More in future architecture classes....

“Hello World!” in Simplified Machine Language

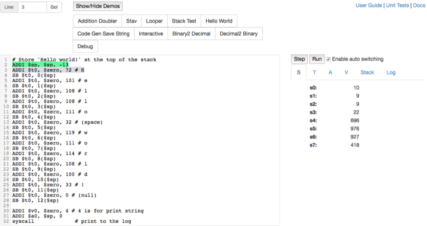


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

s0:	10
s1:	9
s2:	9
s3:	22
s4:	696
s5:	976
s6:	927
s7:	418

(WeMIPS)

WeMIPS



Line: 3 Go!

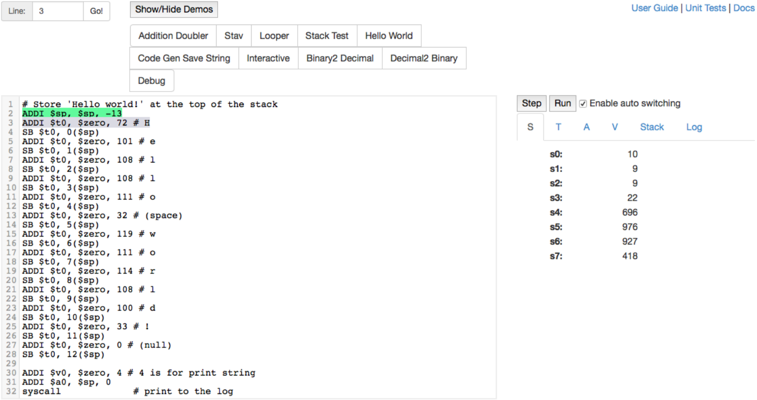
☑ Enable auto switching

```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

```
s0:    10
s1:     9
s2:     9
s3:    22
s4:   696
s5:   976
s6:   927
s7:   418
```

(Demo with WeMIPS)

In Pairs or Triples:

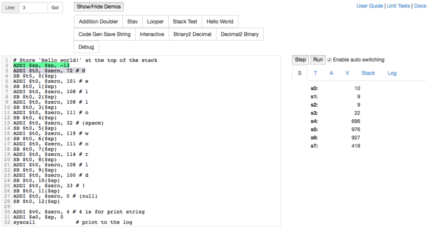


```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

s0:	10
s1:	9
s2:	9
s3:	22
s4:	696
s5:	976
s6:	927
s7:	418

Write a program that prints out the alphabet: `a b c d ... x y z`

WeMIPS



Line: 3

☑ Enable auto switching

```
# Store 'Hello world!' at the top of the stack
ADDI $sp, $sp, -13
ADDI $t0, $zero, 72 # H
SB $t0, 0($sp)
ADDI $t0, $zero, 101 # e
SB $t0, 1($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 2($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 3($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 4($sp)
ADDI $t0, $zero, 32 # (space)
SB $t0, 5($sp)
ADDI $t0, $zero, 119 # w
SB $t0, 6($sp)
ADDI $t0, $zero, 111 # o
SB $t0, 7($sp)
ADDI $t0, $zero, 114 # r
SB $t0, 8($sp)
ADDI $t0, $zero, 108 # l
SB $t0, 9($sp)
ADDI $t0, $zero, 100 # d
SB $t0, 10($sp)
ADDI $t0, $zero, 33 # !
SB $t0, 11($sp)
ADDI $t0, $zero, 0 # (null)
SB $t0, 12($sp)

ADDI $v0, $zero, 4 # 4 is for print string
ADDI $a0, $sp, 0
syscall             # print to the log
```

```
s0:    10
s1:     9
s2:     9
s3:    22
s4:   696
s5:   976
s6:   927
s7:   418
```

(Demo with WeMIPS)

Python & Circuits Review: 10 Weeks in 10 Minutes

A whirlwind tour of the semester, so far...

Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name:  Thomas Hunter
#Date:  September 1, 2017
#This program prints:  Hello, World!

print("Hello, World!")
```

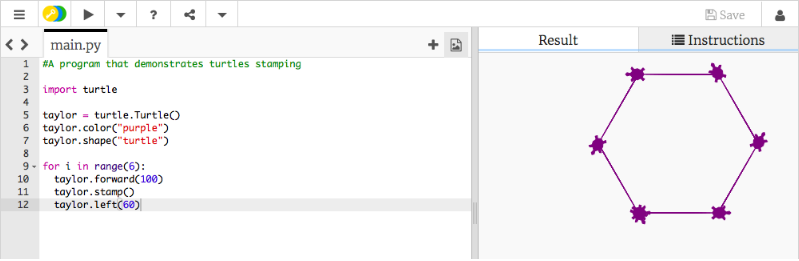
← *These lines are comments*

← *(for us, not computer to read)*

← *(this one also)*

← *Prints the string "Hello, World!" to the screen*

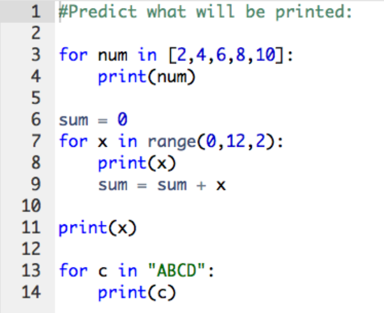
- As well as definite loops & the turtle package:



```
#A program that demonstrates turtles stamping

import turtle

taylor = turtle.Turtle()
taylor.color("purple")
taylor.shape("turtle")

for i in range(6):
  taylor.forward(100)
  taylor.stamp()
  taylor.left(60)
```

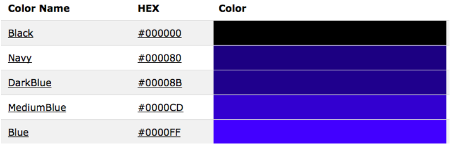
Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - **int**: integer or whole numbers
  - **float**: floating point or real numbers
  - **string**: sequence of characters
  - **list**: a sequence of items
    e.g. `[3, 1, 4, 5, 9]` or `[’violet’,’purple’,’indigo’]`
  - **class variables**: for complex objects, like turtles.
- More on loops & ranges:

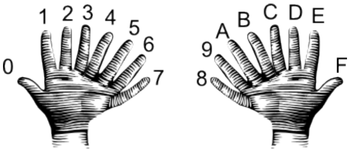


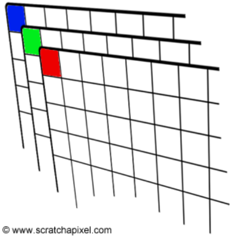
```
#Predict what will be printed:

for num in [2,4,6,8,10]:
    print(num)

sum = 0
for x in range(0,12,2):
    print(x)
    sum = sum + x

print(x)

for c in "ABCD":
    print(c)
```

Week 3: colors, hex, slices, numpy & images

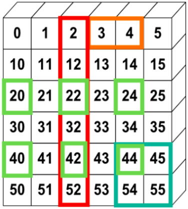


Color Name	HEX	Color
Black	#000000	⬛
Navy	#000080	🟦
DarkBlue	#00008B	🟦
MediumBlue	#0000CD	🟦
Blue	#0000FF	🟦



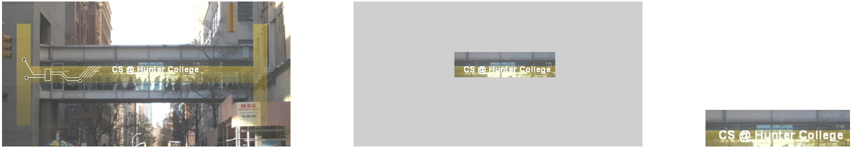

© www.scratchapixel.com

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



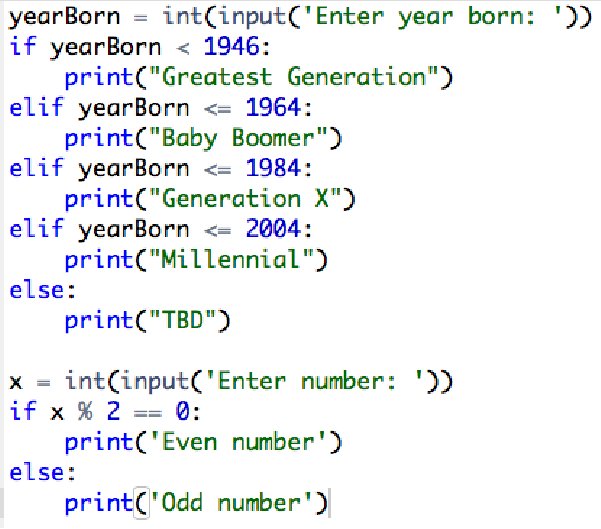
0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

Week 4: design problem (cropping images) & decisions



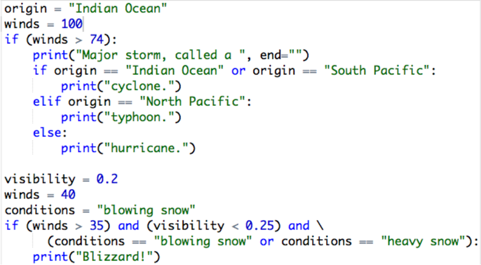
- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
- Next: write pseudocode.
  1. Import numpy and pyplot.
  2. Ask user for file names and dimensions for cropping.
  3. Save input file to an array.
  4. Copy the cropped portion to a new array.
  5. Save the new array to the output file.
- Next: translate to Python.

Week 4: design problem (cropping images) & decisions



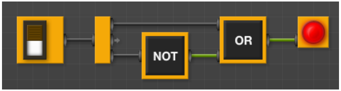
```
yearBorn = int(input('Enter year born: '))
if yearBorn < 1946:
    print("Greatest Generation")
elif yearBorn <= 1964:
    print("Baby Boomer")
elif yearBorn <= 1984:
    print("Generation X")
elif yearBorn <= 2004:
    print("Millennial")
else:
    print("TBD")

x = int(input('Enter number: '))
if x % 2 == 0:
    print('Even number')
else:
    print('Odd number')
```

Week 5: logical operators, truth tables & logical circuits



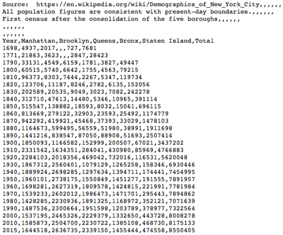
```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
      (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

in1		in2	*returns:*
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True



Week 6: structured data, pandas, & more design

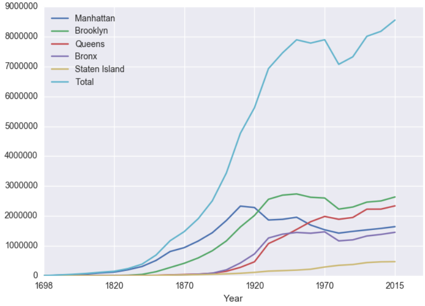
```
import matplotlib.pyplot as plt
import pandas as pd

pop = pd.read_csv(’nycHistPop.csv’,skiprows=5)

pop.plot(x="Year")
plt.show()
```



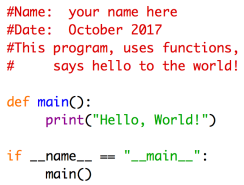
```
Source:  https://en.wikipedia.org/wiki/Demographics_of_New_York_City,,,,,,
All population figures are consistent with present-day boundaries.,,,,,,
First census after the consolidation of the five boroughs,,,,,,
,,,,,,
,,,,,,
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,4549,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,8303,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,5346,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813669,279122,32903,23593,25492,1174779
1870,942292,419921,45468,37393,33029,1478103
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51693,2507414
1900,1850093,1166582,152999,200507,67021,3437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018356,469042,732016,116531,5620048
1930,1867312,2560401,1079129,1265258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1960101,2738175,1550849,1451277,191555,7891957
1960,1698281,2627319,1809578,1424815,221991,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1891325,1168972,352121,7071639
1990,1487536,2300664,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1585873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339150,1455444,474558,8550405
```

`nycHistPop.csv`

In Lab 6



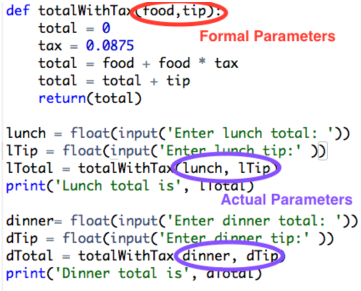
Week 7: functions



```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#     says hello to the world!

def main():
     print("Hello, World!")

if __name__ == "__main__":
     main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis: Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

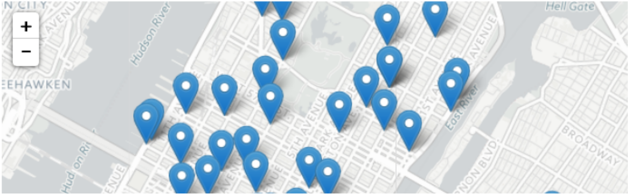
Week 8: function parameters, github

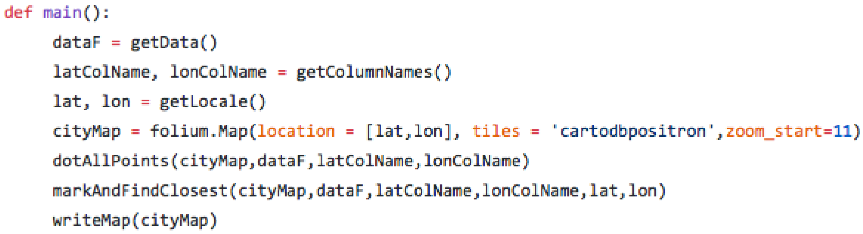


```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip:' ))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip:' ))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The "placeholders" in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.
- Functions can also **return values** to where it was called.

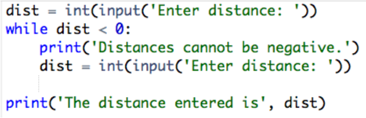
Week 9: top-down design, folium



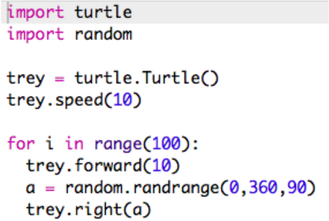


```
def main():
     dataF = getData()
     latColName, lonColName = getColumnNames()
     lat, lon = getLocale()
     cityMap = folium.Map(location = [lat,lon], tiles = 'cartodbpositron',zoom_start=11)
     dotAllPoints(cityMap,dataF,latColName,lonColName)
     markAndFindClosest(cityMap,dataF,latColName,lonColName,lat,lon)
     writeMap(cityMap)
```

Week 10: indefinite loops, searching data, random()



```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```



```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
  trey.forward(10)
  a = random.randrange(0,360,90)
  trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
- To use, must include: `import random`.

Python & Circuits Review: 10 Weeks in 10 Minutes

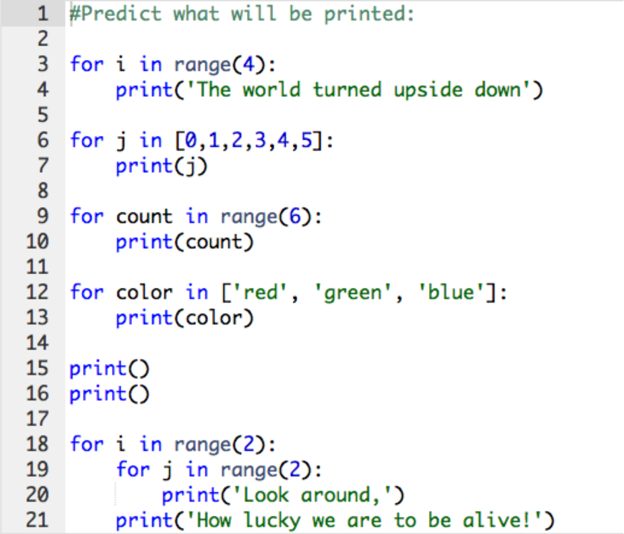
- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
  - Primitive: int, float, bool, string;
  - Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: if-elif-else
- Logical Expressions & Circuits
- Functions: parameters & returns
- Packages:
  - Built-in: turtle, math, random
  - Popular: numpy, matplotlib, pandas, folium

Python & Circuits Review: 10 Weeks in 10 Minutes

A whirlwind tour with
10 (or so) challenges...

In Pairs or Triples: Week 1

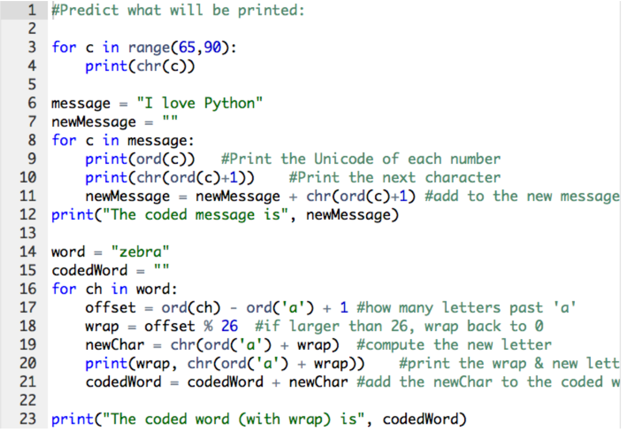
Predict what the code will do:



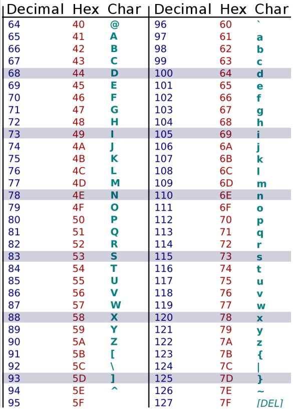
```python
#Predict what will be printed:

for i in range(4):
    print('The world turned upside down')

for j in [0,1,2,3,4,5]:
    print(j)

for count in range(6):
    print(count)

for color in ['red', 'green', 'blue']:
    print(color)

print()
print()

for i in range(2):
    for j in range(2):
        print('Look around,')
    print('How lucky we are to be alive!')
```

In Pairs or Triples: Week 2

Predict what the code will do:



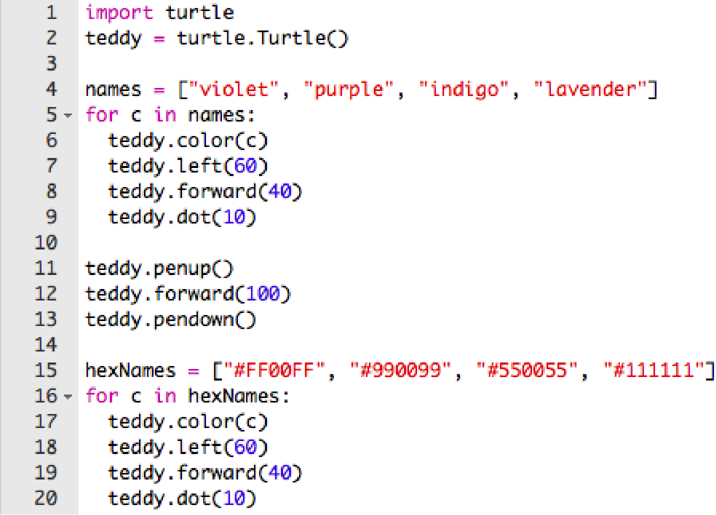
```
#Predict what will be printed:

for c in range(65,90):
    print(chr(c))

message = "I love Python"
newMessage = ""
for c in message:
    print(ord(c))   #Print the Unicode of each number
    print(chr(ord(c)+1))    #Print the next character
    newMessage = newMessage + chr(ord(c)+1) #add to the new message
print("The coded message is", newMessage)

word = "zebra"
codedWord = ""
for ch in word:
    offset = ord(ch) - ord('a') + 1 #how many letters past 'a'
    wrap = offset % 26  #if larger than 26, wrap back to 0
    newChar = chr(ord('a') + wrap)  #compute the new letter
    print(wrap, chr(ord('a') + wrap))    #print the wrap & new lett
    codedWord = codedWord + newChar #add the newChar to the coded w

print("The coded word (with wrap) is", codedWord)
```



Decimal	Hex	Char	Decimal	Hex	Char
64	40	@	96	60	`
65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q
82	52	R	114	72	r
83	53	S	115	73	s
84	54	T	116	74	t
85	55	U	117	75	u
86	56	V	118	76	v
87	57	W	119	77	w
88	58	X	120	78	x
89	59	Y	121	79	y
90	5A	Z	122	7A	z
91	5B	[	123	7B	{
92	5C	\	124	7C	|
93	5D	]	125	7D	}
94	5E	^	126	7E	~
95	5F	_	127	7F	*[DEL]*

In Pairs or Triples: Week 3

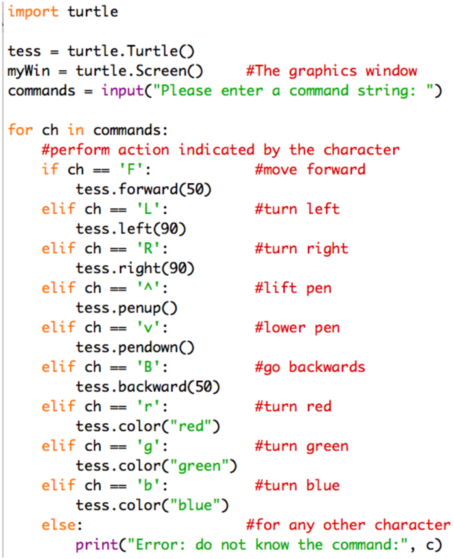
Predict what the code will do:



```python
import turtle
teddy = turtle.Turtle()

names = ["violet", "purple", "indigo", "lavender"]
for c in names:
  teddy.color(c)
  teddy.left(60)
  teddy.forward(40)
  teddy.dot(10)

teddy.penup()
teddy.forward(100)
teddy.pendown()

hexNames = ["#FF00FF", "#990099", "#550055", "#111111"]
for c in hexNames:
  teddy.color(c)
  teddy.left(60)
  teddy.forward(40)
  teddy.dot(10)
```

In Pairs or Triples: Week 4

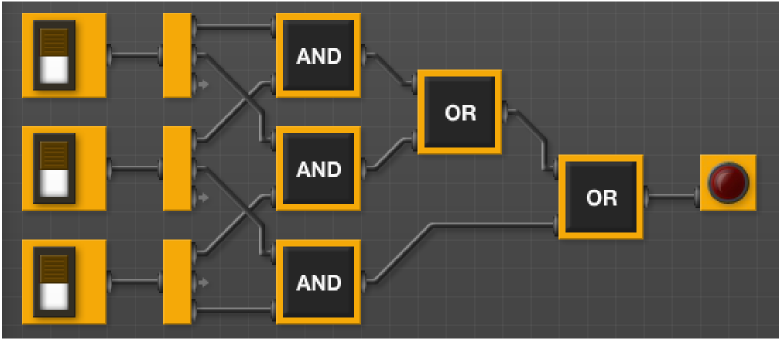
Extend this program to also allow drawing in purple & stamping:



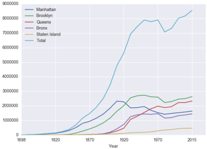
```
import turtle

tess = turtle.Turtle()
myWin = turtle.Screen()      #The graphics window
commands = input("Please enter a command string: ")

for ch in commands:
    #perform action indicated by the character
    if ch == 'F':             #move forward
        tess.forward(50)
    elif ch == 'L':           #turn left
        tess.left(90)
    elif ch == 'R':           #turn right
        tess.right(90)
    elif ch == '^':           #lift pen
        tess.penup()
    elif ch == 'v':           #lower pen
        tess.pendown()
    elif ch == 'B':           #go backwards
        tess.backward(50)
    elif ch == 'r':           #turn red
        tess.color("red")
    elif ch == 'g':           #turn green
        tess.color("green")
    elif ch == 'b':           #turn blue
        tess.color("blue")
    else:                    #for any other character
        print("Error: do not know the command:", c)
```

In Pairs or Triples: Week 5

When does this circuit yield true?
That is, what values for the inputs give an output value of true?



In Pairs or Triples: Week 6



Predict what the following will do:

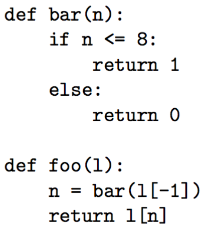
- `print("Queens:", pop["Queens"].min())`
- `print("S I:", pop["Staten Island"].mean())`
- `print("S I:", pop["Staten Island"].std())`
- `pop.plot.bar(x="Year")`
- `pop.plot.scatter(x="Brooklyn", y= "Total")`
- `pop["Fraction"] = pop["Bronx"]/pop["Total"]`

In Pairs or Triples: Week 7

Fill in the function body:

```python
def monthString(monthNum):
    """
    Takes as input a number, monthNum, and
    returns the corresponding month name as a string.
    Example: monthString(1) returns "January".
    Assumes that input is an integer ranging from 1 to 12
    """

    monthString = ""

    ###################################
    ### FILL IN YOUR CODE HERE      ###
    ### Other than your name above, ###
    ### this is the only section    ###
    ### you change in this program. ###
    ###################################

    return(monthString)

def main():
    n = int(input('Enter the number of the month: '))
    mString = monthString(n)
    print('The month is', mString)
```

In Pairs or Triples: Week 8



```
def bar(n):
    if n <= 8:
        return 1
    else:
        return 0

def foo(l):
    n = bar(l[-1])
    return l[n]
```

- What are the formal parameters for the functions?
- What is the output of:

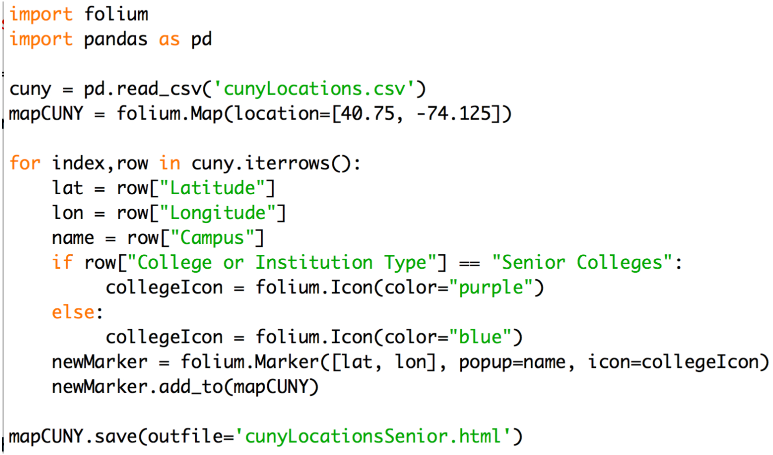
  ```
  r = foo([1,2,3,4])
  print("Return:  ", r)
  ```

- What is the output of:

  ```
  r = foo([1024,512,256,128])
  print("Return:  ", r)
  ```

In Pairs or Triples: Week 9

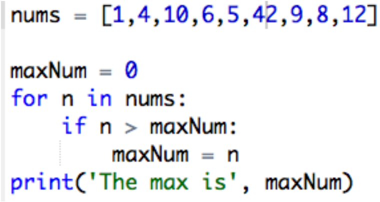
What does this code do?



```python
import folium
import pandas as pd

cuny = pd.read_csv('cunyLocations.csv')
mapCUNY = folium.Map(location=[40.75, -74.125])

for index,row in cuny.iterrows():
    lat = row["Latitude"]
    lon = row["Longitude"]
    name = row["Campus"]
    if row["College or Institution Type"] == "Senior Colleges":
        collegeIcon = folium.Icon(color="purple")
    else:
        collegeIcon = folium.Icon(color="blue")
    newMarker = folium.Marker([lat, lon], popup=name, icon=collegeIcon)
    newMarker.add_to(mapCUNY)

mapCUNY.save(outfile='cunyLocationsSenior.html')
```

In Pairs or Triples: Week 10

- *Predict what the code will do:*



```
nums = [1,4,10,6,5,42,9,8,12]

maxNum = 0
for n in nums:
    if n > maxNum:
        maxNum = n
print('The max is', maxNum)
```

- Write a function that asks a user for number after 2000 but before 2018. The function should repeatedly ask the user for a number until they enter one within the range and return the number.

Python & Circuits Review: 10 Weeks in 10 Minutes

- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
  - Primitive: int, float, bool, string;
  - Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: if-elif-else
- Logical Expressions & Circuits
- Functions: parameters & returns
- Packages:
  - Built-in: turtle, math, random
  - Popular: numpy, matplotlib, pandas, folium

Final Overview: Format

- The exam is 2 hours long.
- There are 4 different versions to discourage copying.
- It is on paper. No use of computers, phones, etc. allowed.
- You may have 1 piece of **8.5" x 11"** piece of paper.
  - With notes, examples, programs: what will help you on the exam.
  - No origami– it's distracting to others taking the exam.
  - Best if you design/write yours since excellent way to study.
- The exam format:
  - 10 questions, each worth 10 points.
  - Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.
- Past exams available on webpage (includes answer keys).

Recap: Python, Languages, & Design

```
#Name:  your name here
#Date:  October 2017
#This program, uses functions,
#     says hello to the world!

def main():
     print("Hello, World!")

if __name__ == "__main__":
     main()
```

- On lecture slip, write down a topic you wish we had spent more time (and why).
- Python language
- Logical Circuits
- Simplified Machine Language
- Design: from written description (‘specs’) to function inputs & outputs (‘APIs’)

Final Overview: Top-Down Design & APIs

For each question, write **only the function header (name & inputs) and return values** (often called the Application Programming Interface (API)):

- Write a function that takes a weight in kilograms and returns the weight in pounds.
- Write a function that takes a string and returns its length.
- Write a function that, given a DataFrame, returns the minimal value in the first column.
- Write a function that takes a whole number and returns the corresponding binary number as a string.
- Write a function that computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.

(Hint: highlight key words, make list of inputs, list of outputs, then put together.)

Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a weight in kilograms and returns the weight in pounds.

```
def kg2lbs(kg)
    lbs = kg * 2.2
    return(lbs)
```

Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a string and returns its length.

Final Overview

For each question below, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that, given a DataFrame, returns the minimal value in the "Manhattan" column.

```
def getMin(df):
    mM = df['Manhattan'].min()
    return(mM)
```

Final Overview

For each question, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that takes a whole number and returns the corresponding binary number as a string.

```
def num2bin(num):
    binStr = ""
    while (num > 0):
        #Divide by 2, and add the remainder to the string
        r = num %2
        binString = str(r) + binStr
        num = num / 2
    return(binStr)
```

Final Overview

For each question below, write the function header (name & inputs) and return values (often called the Application Programming Interface (API)):

- Write a function that computes the total monthly payment when given the initial loan amount, annual interest rate, number of years of the loan.

```
def computePayment(loan,rate,year):
    (Some formula for payment)
    return(payment)
```