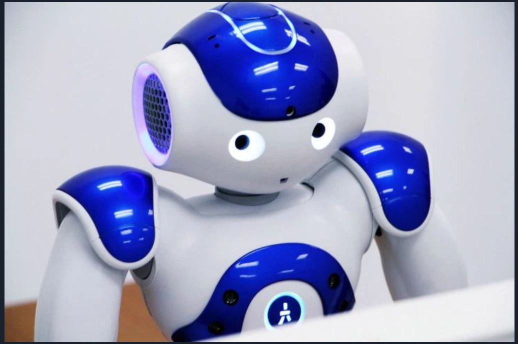
Welcome to Python!

Ryan Sander, Srimayi Tenali, Megan Ochalek, Alex Miller

INJAZ 2019

Why Learn to Code?

- Improved productivity
- Practice problem-solving skills
- Many career opportunities:
    - Algorithms
    - AI/Robotics
    - Data Science
    - Web Development
    - Embedded Programming
    - Signal Processing
- A lot of fun!



Why Learn to Code?

Lesson topics:

- Intro to Python, and Syntax
- Conditional Logic and Data Structures
- Loops and Iteration
- Functions
- Prob/Stat + Python!

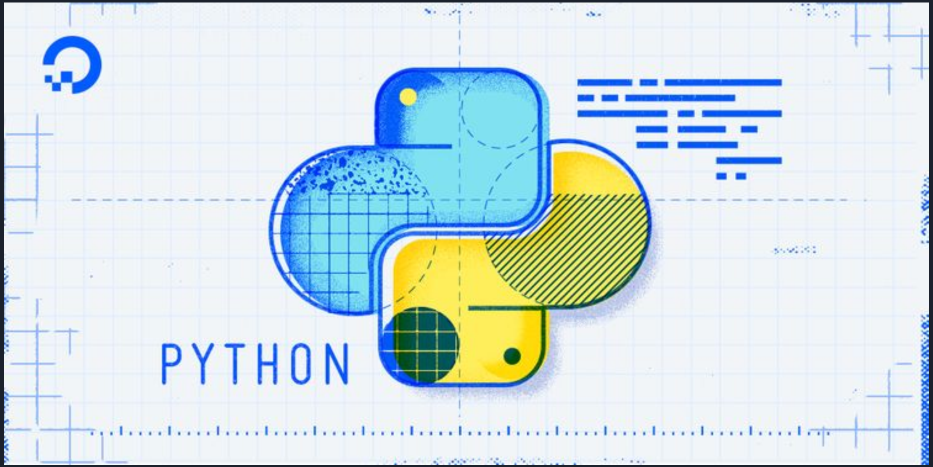
Key takeaways:

- Problem solving using programming
- Introduction to Computer Science
- Computational thinking
- Understand how computers process information

Intro to Python, and Syntax!

Questions we will answer today:

- What is Python?
- How do we write a program?
- How do we read a Python program?



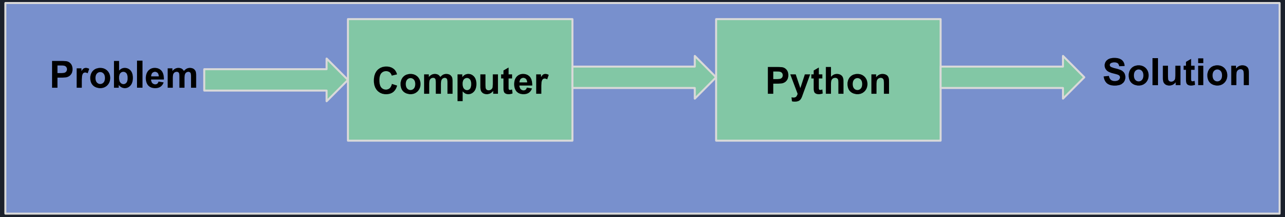
What is Python?

From Wikipedia:

“Python is an interpreted, high-level, general-purpose programming language.”

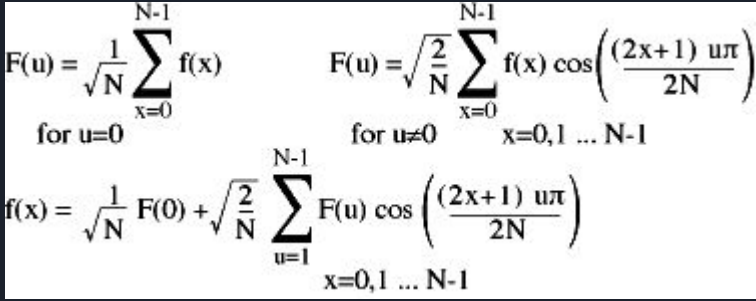
What Does This Mean in English?

- Python is a tool you can use to solve complex problems with computers!
- Python is used to do mathematical operations on numbers that are simply too large for humans to compute.



What is a Program?

- A program is a set of instructions that specifies how to perform a computation.
- Examples of computations:
  - Solving a complicated math problem.
  - Finding the shortest path between two cities.
  - Encoding and decoding a secret message.

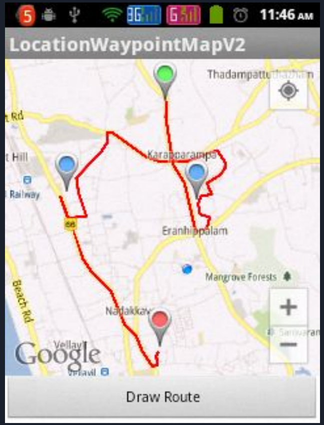


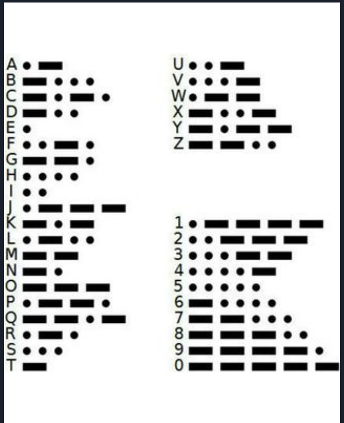
$$F(u) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} f(x) \qquad F(u) = \sqrt{\frac{2}{N}} \sum_{x=0}^{N-1} f(x) \cos\left(\frac{(2x+1)\, u\pi}{2N}\right)$$

for u=0 for u≠0 x=0,1 ... N-1

$$f(x) = \frac{1}{\sqrt{N}} F(0) + \sqrt{\frac{2}{N}} \sum_{u=1}^{N-1} F(u) \cos\left(\frac{(2x+1)\, u\pi}{2N}\right)$$

x=0,1 ... N-1





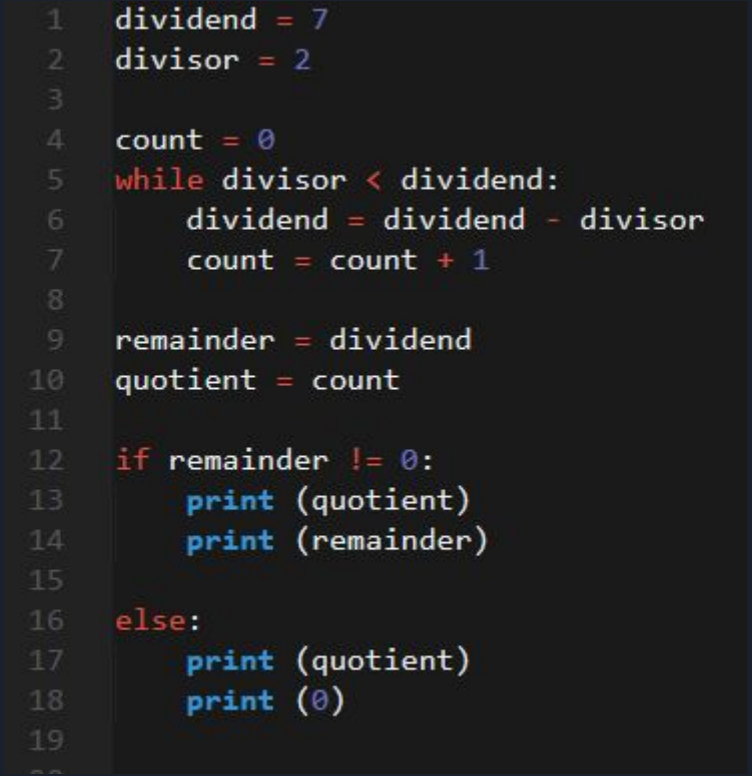
A	•—
B	—•••
C	—•—•
D	—••
E	•
F	••—•
G	——•
H	••••
I	••
J	•———
K	—•—
L	•—••
M	——
N	—•
O	———
P	•——•
Q	——•—
R	•—•
S	•••
T	—
U	••—
V	•••—
W	•——
X	—••—
Y	—•——
Z	——••
1	•————
2	••———
3	•••——
4	••••—
5	•••••
6	—••••
7	——•••
8	———••
9	————•
0	—————

What’s in a Program?

- **input:** Get data from the keyboard, a file, the network, or some other device.
- **output:** Display data on the screen, save it in a file, send it over the network.
- **math:** Perform basic mathematical operations like addition and multiplication.
- **conditional logic:** Only run code under certain conditions.
- **repetition:** Perform some action repeatedly, usually with some variation.

Examples of Programs

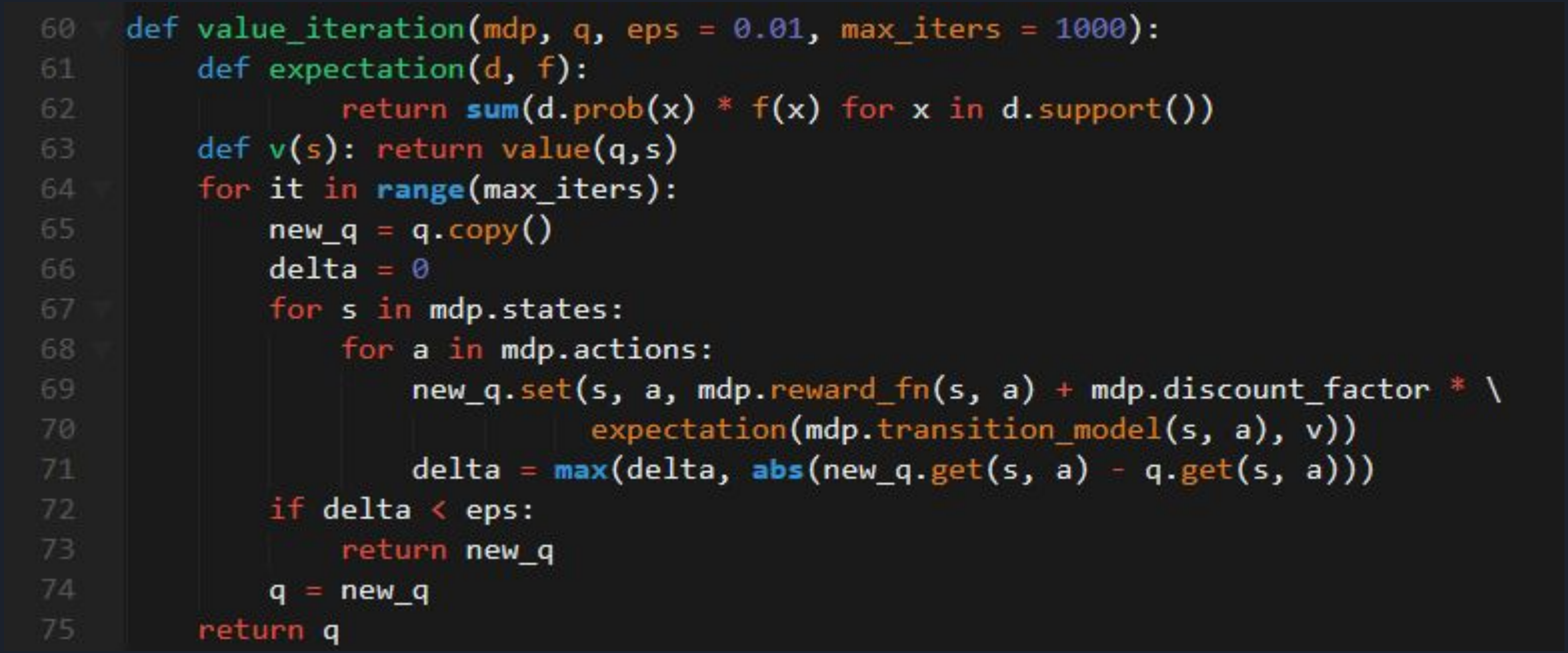
The program to the right can be used to find the remainder for division!



```
dividend = 7
divisor = 2

count = 0
while divisor < dividend:
    dividend = dividend - divisor
    count = count + 1

remainder = dividend
quotient = count

if remainder != 0:
    print (quotient)
    print (remainder)

else:
    print (quotient)
    print (0)

```

Examples of Programs

The program below can be used to help a robot plan a path through a maze!



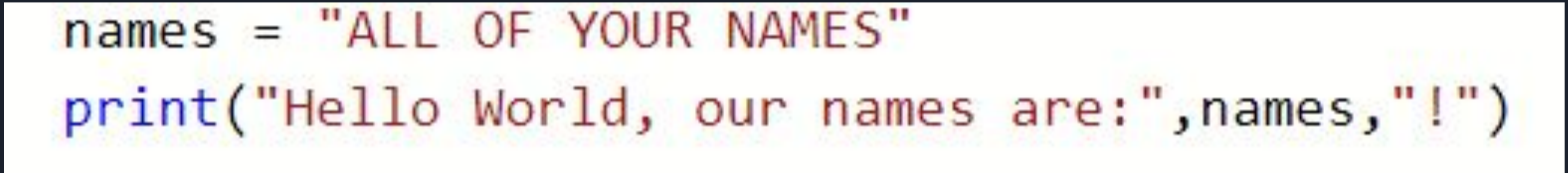
```
def value_iteration(mdp, q, eps = 0.01, max_iters = 1000):
    def expectation(d, f):
            return sum(d.prob(x) * f(x) for x in d.support())
    def v(s): return value(q,s)
    for it in range(max_iters):
        new_q = q.copy()
        delta = 0
        for s in mdp.states:
            for a in mdp.actions:
                new_q.set(s, a, mdp.reward_fn(s, a) + mdp.discount_factor * \
                          expectation(mdp.transition_model(s, a), v))
                delta = max(delta, abs(new_q.get(s, a) - q.get(s, a)))
        if delta < eps:
            return new_q
        q = new_q
    return q
```

Your First Program!

- This will be our first exercise as a class! Please get into groups around each laptop.
- In your groups of five, use your laptops, open your internet browser, and type this in the address bar:
- **https://tinyurl.com/INJAZ-1**

Your First Program!

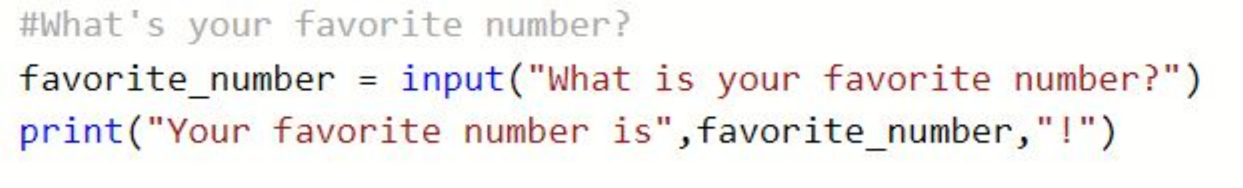
- Please type the following into your IDLE window, exactly like it is below but with your own names:



```python
names = "ALL OF YOUR NAMES"
print("Hello World, our names are:",names,"!")
```

Print and Input Statements

- We will frequently be using two of Python's built-in functions*
  - print(): displays whatever you put inside the parentheses!
  - input(): asks the user for input using what's inside the parentheses!
- To display our inputs and outputs on our computer!

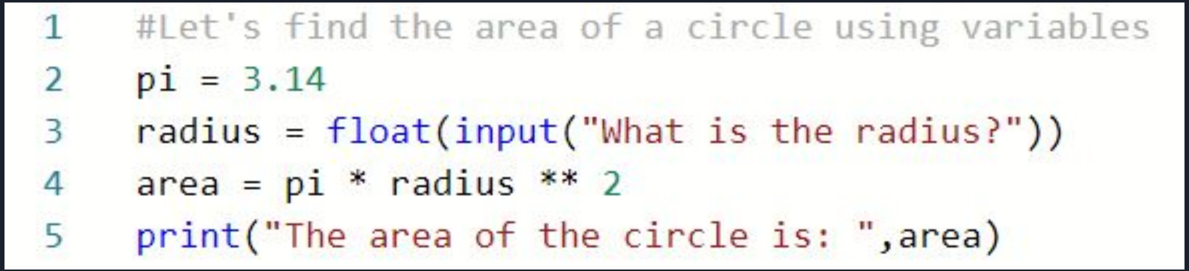


```
#What's your favorite number?
favorite_number = input("What is your favorite number?")
print("Your favorite number is",favorite_number,"!")
```

*We'll cover functions in lesson 4!

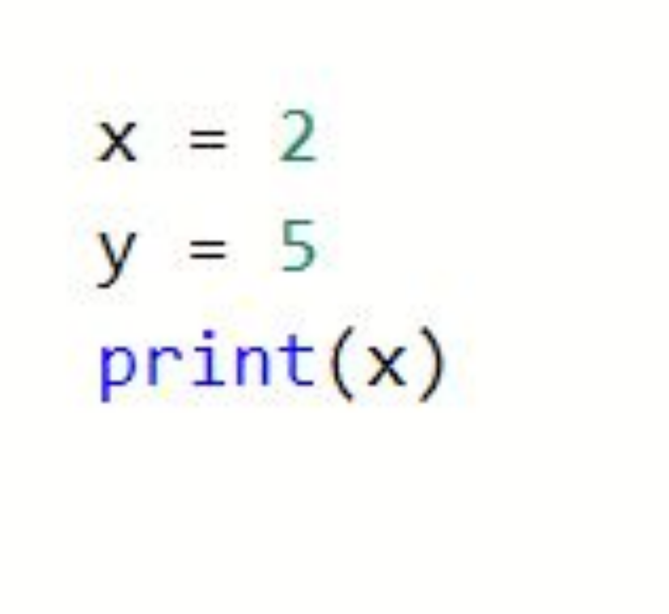
How Do We Store Information in Variables?

- You can store useful information with variables!
  - Nearly every program must store information.
  - Information that is being saved might be user input, names, values.
  - This is called ‘assigning’ a value to a variable.



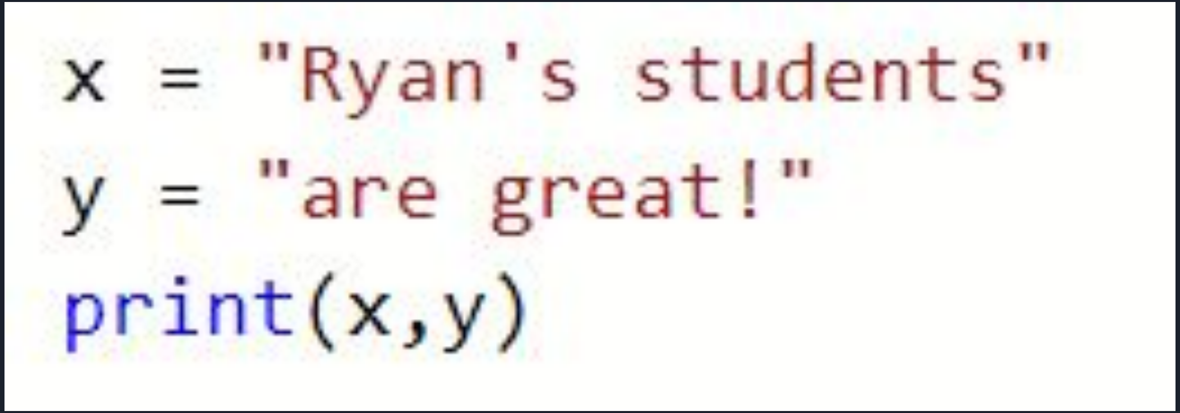
```python
#Let's find the area of a circle using variables
pi = 3.14
radius = float(input("What is the radius?"))
area = pi * radius ** 2
print("The area of the circle is: ",area)
```

What Will This Print?



```
x = 2
y = 5
print(x)
```

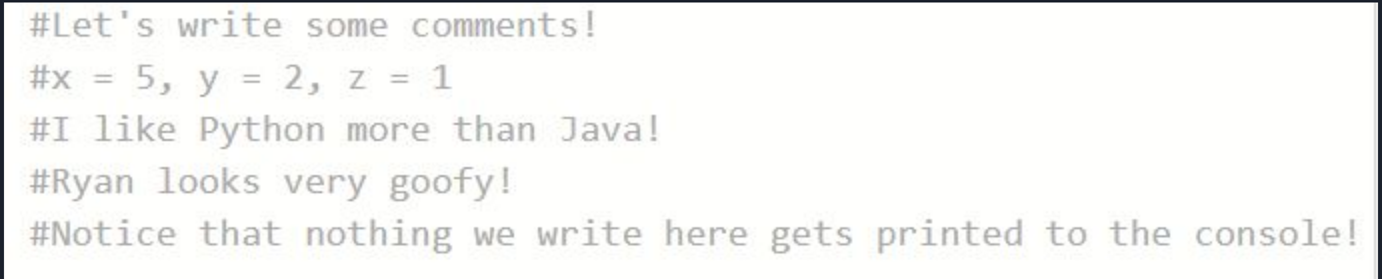
What Will This Print?



```
x = "Ryan's students"
y = "are great!"
print(x,y)
```

What If We Want to Add Notes To Our Code?

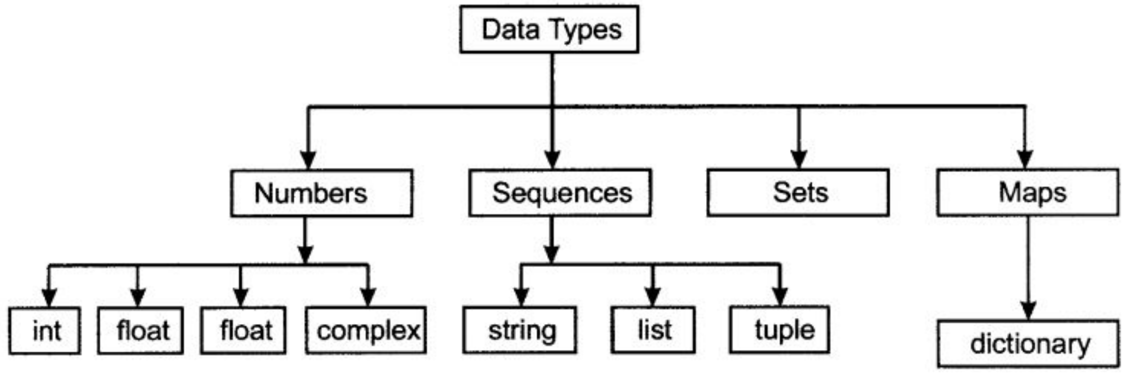
- Comments are for **documenting** our code!
-
- Use ‘#’ (Shift + 3) on your keyboard to begin a new comment!



```
#Let's write some comments!
#x = 5, y = 2, z = 1
#I like Python more than Java!
#Ryan looks very goofy!
#Notice that nothing we write here gets printed to the console!
```

Data Types

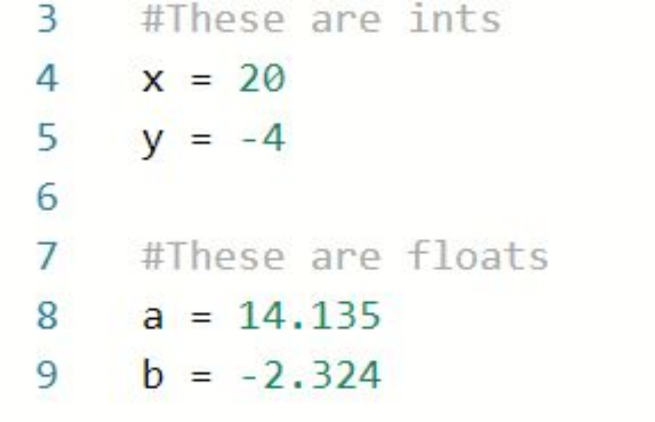
- Numbers
- Strings
- NoneType
- Boolean
- List (Later)
- Tuple (Later)
- Dictionary (Later)



Python Data types

How Do Computers Store Numbers?

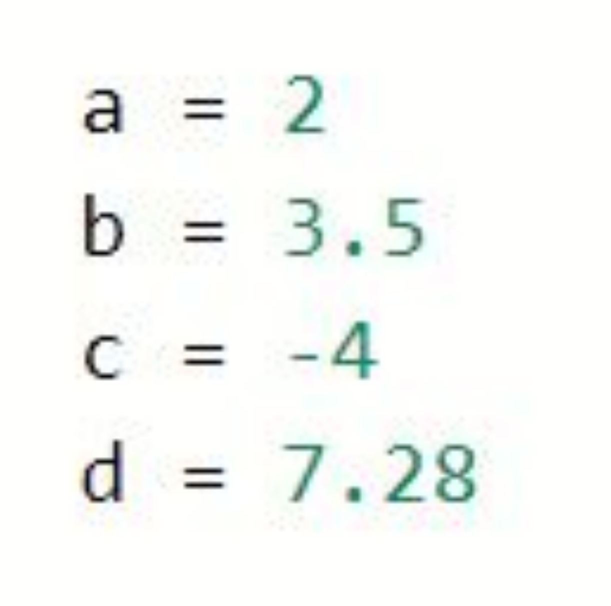
- **Integers** vs **floats**
  - **Int** - Positive or negative **whole numbers**
  - **Float** - Real numbers with **decimals**



```
#These are ints
x = 20
y = -4

#These are floats
a = 14.135
b = -2.324
```

- **Mathematical operations**:
  - int(x): convert x to int type
  - float(x): convert x to float type
  - abs(x): absolute value of x
  - max(x1,x2,x3...): return largest number
  - min(x1,x2,x3,...): return smallest number
  - sqrt(x): square root of x

Int or Float?

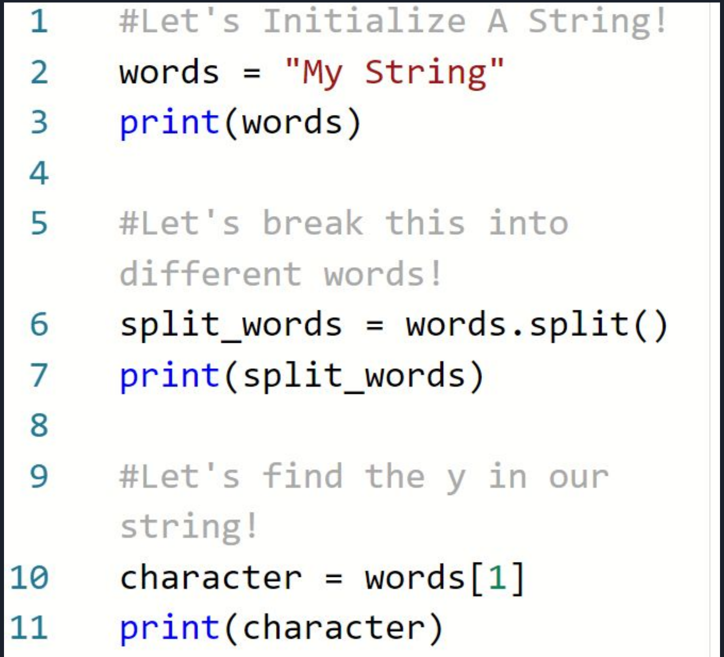


```
a = 2
b = 3.5
c = -4
d = 7.28
```

How Do We Make Words and Sentences In Python?

- Strings are a kind of **sequence.**
- Use parentheses!
- To change data to a string:

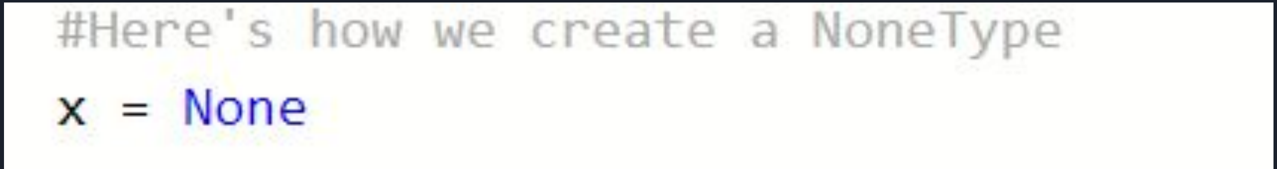
str(x): converts x to a str type



```python
#Let's Initialize A String!
words = "My String"
print(words)

#Let's break this into different words!
split_words = words.split()
print(split_words)

#Let's find the y in our string!
character = words[1]
print(character)
```

What if Our Variable Doesn’t Have a Type?

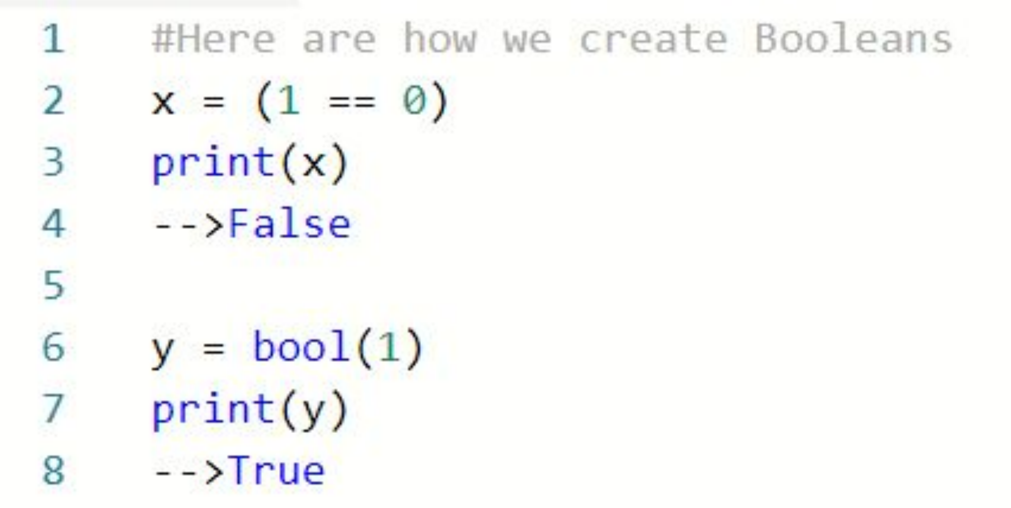
- NoneType variable <--> “No Type”
- Usually comes up when we have errors in our program.



```
#Here's how we create a NoneType
x = None
```

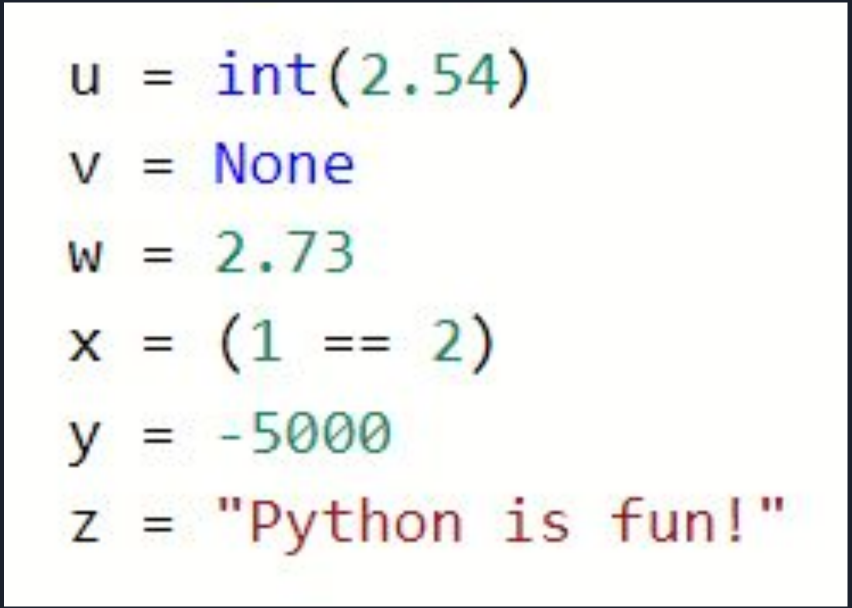
How Do We Use True and False in Python?

- A boolean evaluates to either **True** or **False**. Examples:



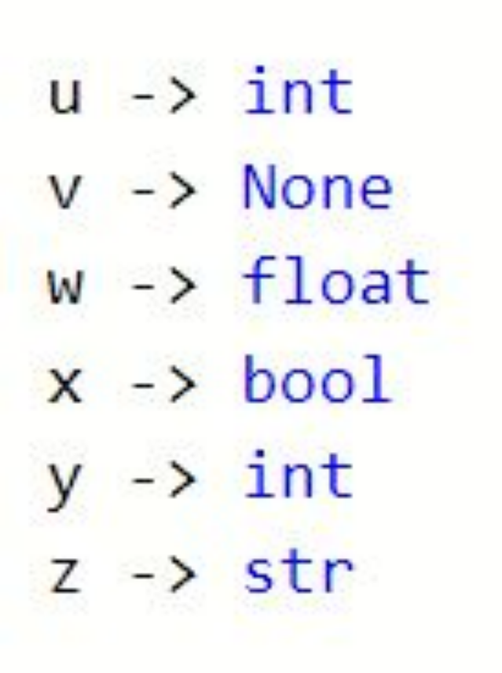
```
#Here are how we create Booleans
x = (1 == 0)
print(x)
-->False

y = bool(1)
print(y)
-->True
```

Name That Type!



```
u = int(2.54)
v = None
w = 2.73
x = (1 == 2)
y = -5000
z = "Python is fun!"
```

Name That Type!



```
u -> int
v -> None
w -> float
x -> bool
y -> int
z -> str
```

Exercise!

Please go to the following:

https://tinyurl.com/name-that-type

How Do We Do Math in Python?

(a=10, b = 20)

Python Operator	**Description**	**Example**
+	Addition	a+b=30
-	Subtraction	a-b=-10
*	Multiplication	a*b=200
/	Division	b/a=2
%	Modulus	b%a=0
**	Exponent	a**b = a to the power of b

Why Use Modulus (%)?

- Figure out if one number is divisible by another!

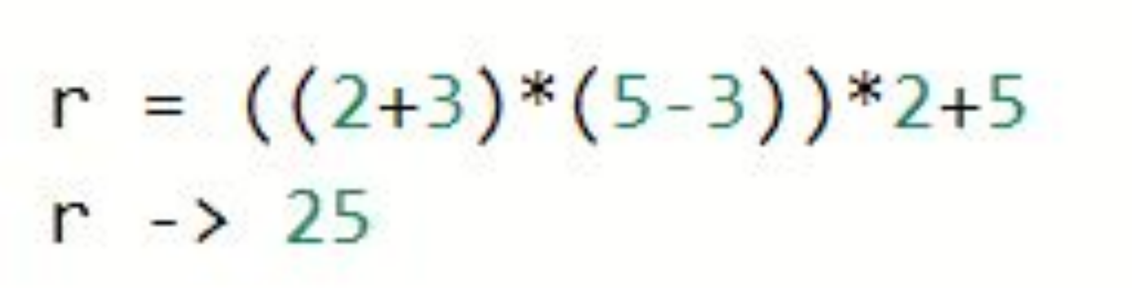
- If a % b = 0, a is divisible by b!

How Do We Compare Variables To Each Other?

Operator	**Description**	**Example**
==	If values are equal, the condition becomes true	(10==20) is not true
!=	If values are not equal, then condition becomes true	(10!=20) is true
>,>=	Greater than, greater than or equal to	(10>20), (10>=20) are not true
<,<=	Less than, less than or equal to	(10<20),(10<=20) are true

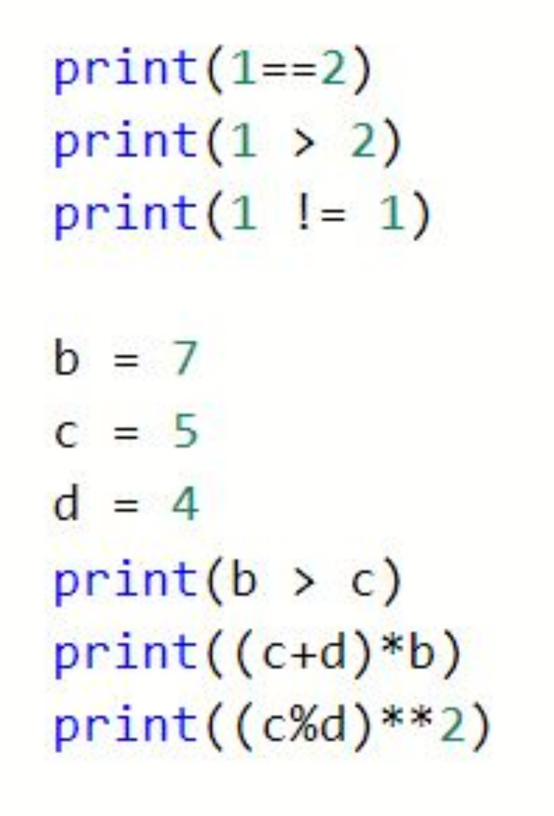
How Do We Know The Order of Operations?

- Python evaluates operations in parentheses before anything else.
- Next comes **, then * and /, and then + and -.



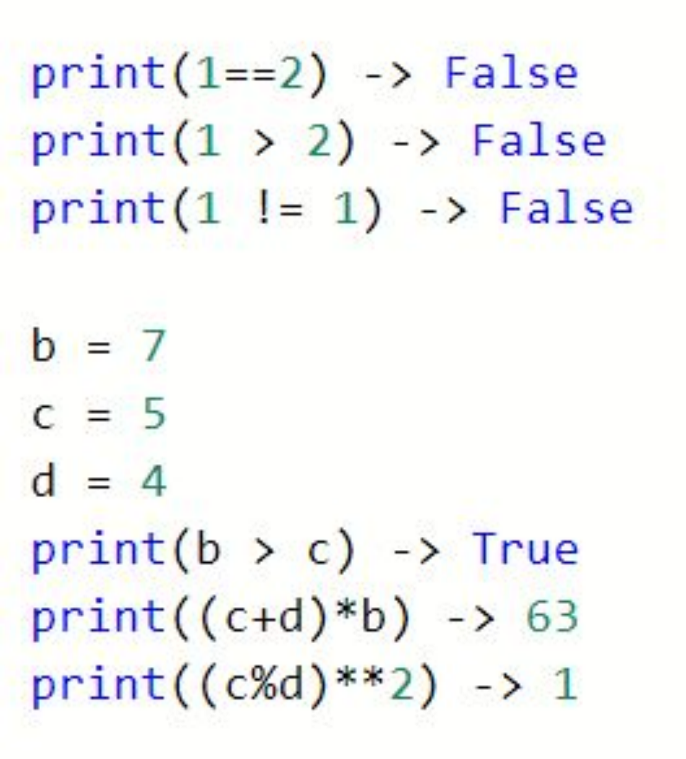
```
r = ((2+3)*(5-3))*2+5
r -> 25
```

Let’s Practice! What Do These Print?



```
print(1==2)
print(1 > 2)
print(1 != 1)

b = 7
c = 5
d = 4
print(b > c)
print((c+d)*b)
print((c%d)**2)
```

Let’s Practice! What Do These Print?



```
print(1==2) -> False
print(1 > 2) -> False
print(1 != 1) -> False

b = 7
c = 5
d = 4
print(b > c) -> True
print((c+d)*b) -> 63
print((c%d)**2) -> 1
```

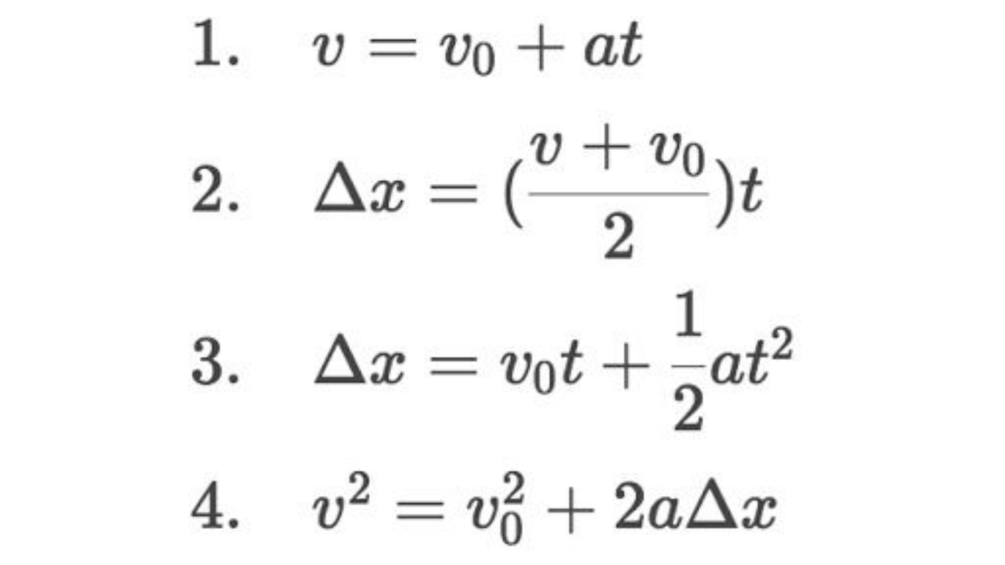
Exercise!

Please go to the following:

https://tinyurl.com/python-maths

Kinematics in Python!

Remember these equations from before?



1. $v = v_0 + at$

2. $\Delta x = (\frac{v + v_0}{2})t$

3. $\Delta x = v_0 t + \frac{1}{2}at^2$

4. $v^2 = v_0^2 + 2a\Delta x$

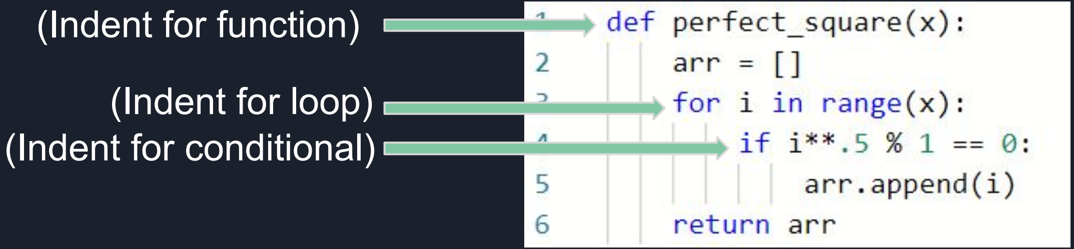
Your turn!

Please go here for some fun exercises!

https://tinyurl.com/python-kinematics

Indenting in Python

- Indents! These are very important in Python. We indent whenever we:
    - Use conditional statements (Lesson 2)
    - Begin a for or while loop (Lesson 4)
    - Define a function (Lesson 5)
- When the indent ends, the part of the program that caused that indent ends too!



(Indent for function)

(Indent for loop)

(Indent for conditional)

```
def perfect_square(x):
    arr = []
    for i in range(x):
        if i**.5 % 1 == 0:
            arr.append(i)
    return arr
```

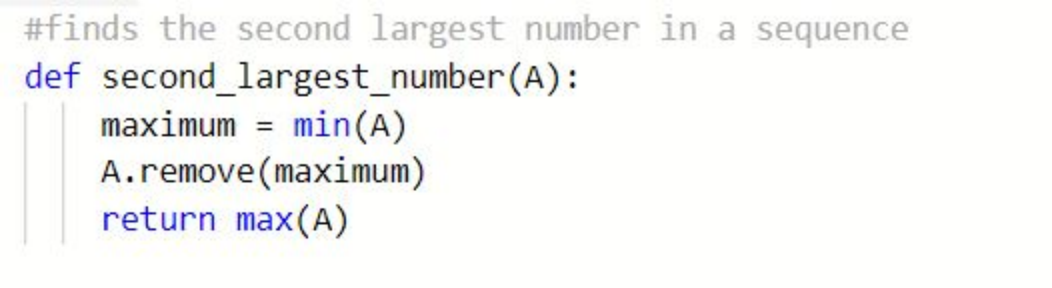
What Do We Do If Our Code Doesn’t Work On the First Try?

- A **bug** is an error in a program! Since our code needs to be exactly correct in order for our program to run, it’s important to always check for bugs!
- When we find a bug, we use a process called **Debugging** to fix our code. Let’s practice!

Debugging in Python

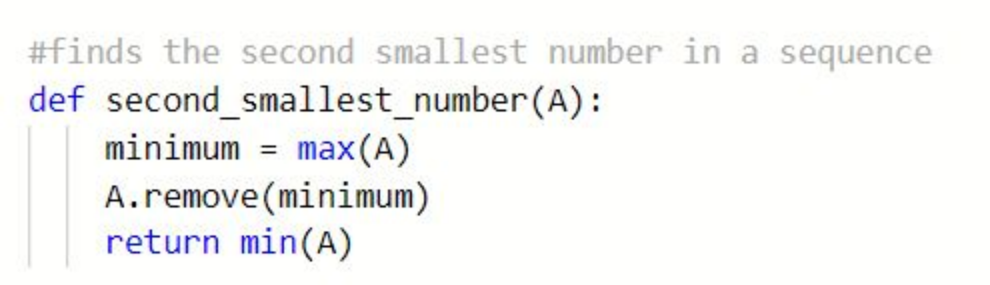
- Try printing variable values at different points in the program!
- If the console gives you an error, read the error! See if you recognize where it could be coming from.
- Divide and conquer! If your program has multiple sections, work on fixing one section at a time.
- Comment your code.
- Ask for help!

Can You Help Me Find the Bug?



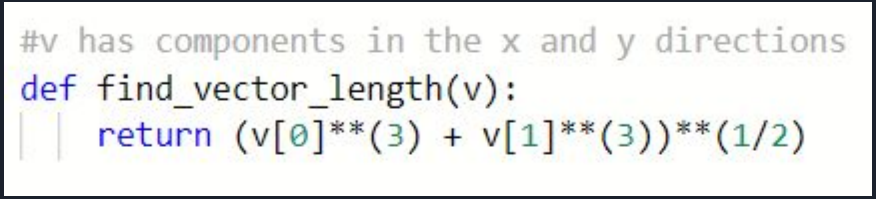
```python
#finds the second largest number in a sequence
def second_largest_number(A):
    maximum = min(A)
    A.remove(maximum)
    return max(A)
```

Can You Help Me Find the Bug?



```
#finds the second smallest number in a sequence
def second_smallest_number(A):
    minimum = max(A)
    A.remove(minimum)
    return min(A)
```

Can You Help Me Find the Bug?



```
#v has components in the x and y directions
def find_vector_length(v):
    return (v[0]**(3) + v[1]**(3))**(1/2)
```

Conditional Logic!

Questions we will answer:

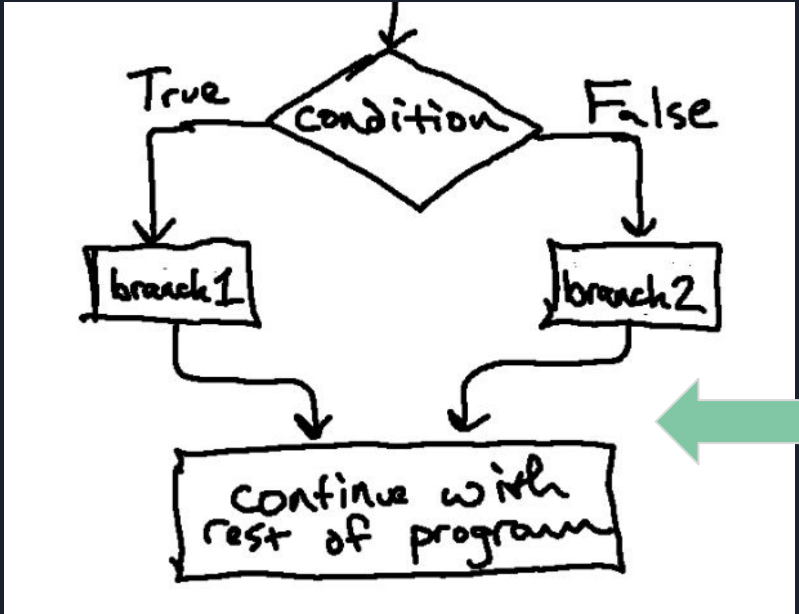
- How do we implement logic in Python?
- How can we tell Python to print something, but only sometimes?
- What are if, elif, and else statements?

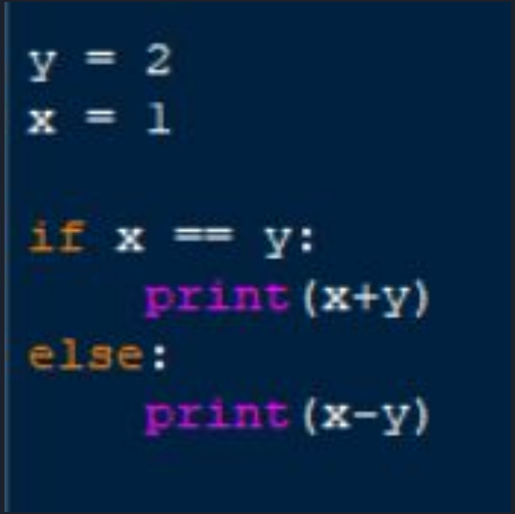
What is Conditional Logic?

- Uses **logical operators** for branching:
  - if
  - elif (known as “else if”)
  - else

Conditional Logic Diagram

- Here is an illustration of conditional logic in a program:

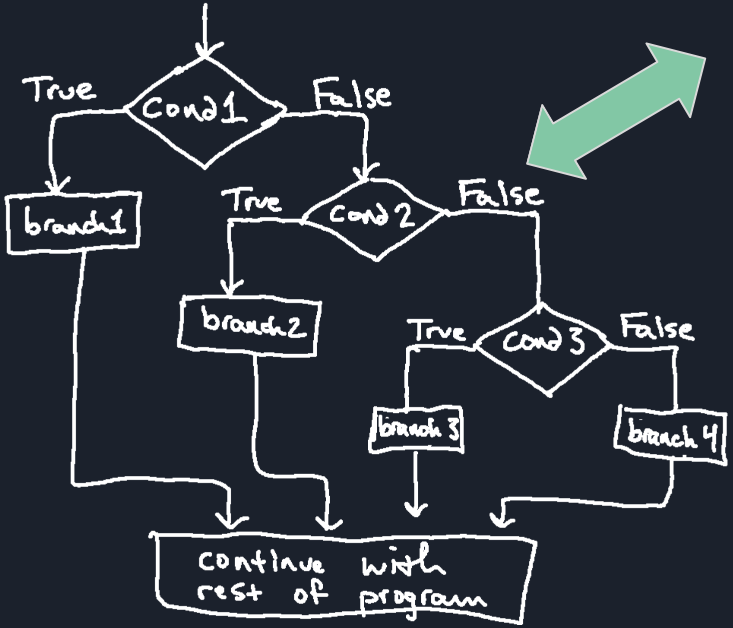


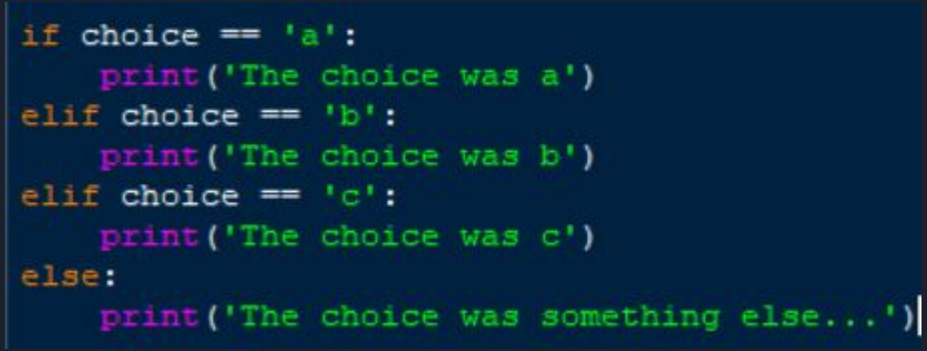


```
y = 2
x = 1

if x == y:
    print(x+y)
else:
    print(x-y)
```

Chained Conditional Logic

- **Elif** statements let us use chained conditional logic.

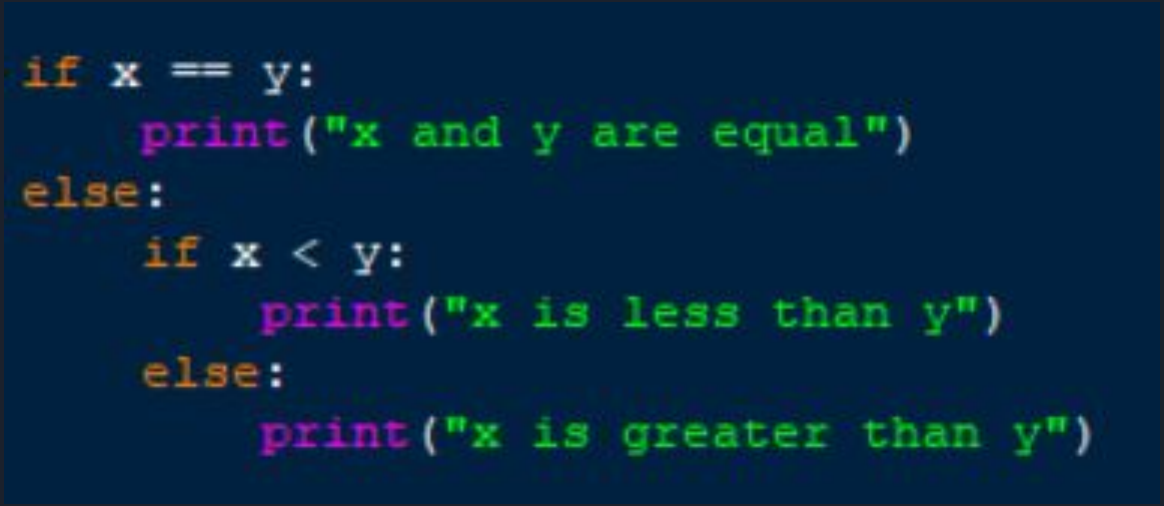




```python
if choice == 'a':
    print('The choice was a')
elif choice == 'b':
    print('The choice was b')
elif choice == 'c':
    print('The choice was c')
else:
    print('The choice was something else...')
```

Nested Conditional Logic

- We can make conditional logic branches branches off of branches!



```
if x == y:
    print("x and y are equal")
else:
    if x < y:
        print("x is less than y")
    else:
        print("x is greater than y")
```

Exercise: Pizza!

https://tinyurl.com/python-pizza-party

Or and And Operators

- A or B: If either A or B is True, then return True. Otherwise, return False.

- A and B: If A and B are both True, then return True. Otherwise, return False.

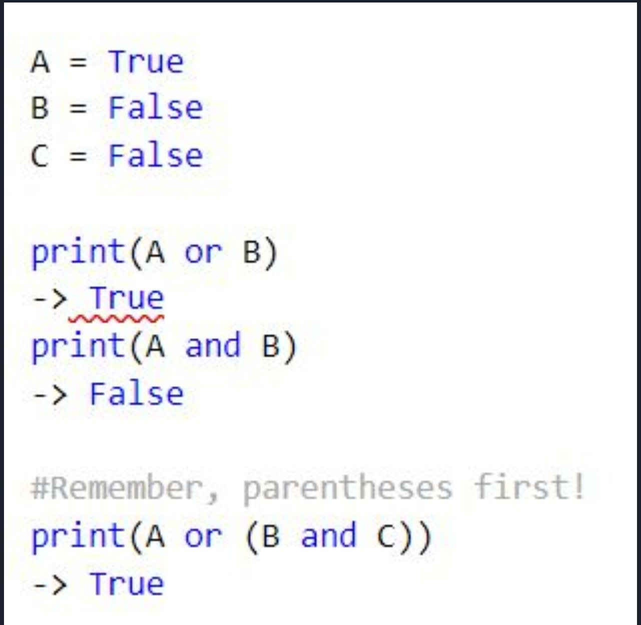
AND Truth Table

A	**B**	**A AND B**
F	F	F
F	T	F
T	F	F
T	T	T

OR Truth Table

A	**B**	**A OR B**
F	F	F
F	T	T
T	F	T
T	T	T

Or and And in Python!



```
A = True
B = False
C = False

print(A or B)
-> True
print(A and B)
-> False

#Remember, parentheses first!
print(A or (B and C))
-> True
```

Exercise!

Please go to the following:

https://tinyurl.com/python-or-and

Your Turn!

1. I want you to write a program that finds out what kind of tea your friend wants!
   a. First, ask someone the kind of tea (hint: use the built-in function **input()**) they'd like.
   b. Now, check to see if they want black tea (hint: use an **if** statement).
      i. If they do, tell your customer "Here's your black tea!"
   c. If they don't want black tea, check if they want green or chai tea (hint: use **elif** statements).
      i. If they do, tell your customer "Here's your green/chai tea!"
   d. Finally, if you don't have the kind of tea your friend wants, tell them (hint: use an **else** statement).

2. Next, we’re going to make a simple calculator!

a. First, ask your friend for two numbers and an arithmetic operator (+,-,*,/) (hint: use the **input()** function three times).

b. Then, check to see if the operator is addition! (hint: use an **if** statement).

   i. If it is, add the two numbers together and print the result.

c. If it isn’t addition, check to see if the operator is subtraction, multiplication, or division! (hint: use three **elif** statements).

   i. If it is, subtract/multiply/divide the first number and/by the second number and print the result.

d. If the operation isn’t one of the ones above, print (“ERROR”) (hint: use an **else** statement).

Challenge Problem!

- Let’s use Python to figure this out!
- Problem: There are 100 doors in a row, numbered 1-100, each of which starts out locked. You make 100 passes through the doors.
  - On the **first** pass, you switch the state of the locks (locked doors become unlocked, and unlocked doors become locked) on doors **1, 2, 3, 4,…., 100**.
  - On the **second** pass, you switch the state of the locks on doors **2, 4, 6, 8,…,100.**
  - On the **third** pass, you switch the state of the locks on doors **3, 6, 9, 12,…,99**, and so on.
  - You do this until you reach 100, at which time you only switch the lock on door 100.
- After 100 passes, which doors will be unlocked?

Warm-up Activity: Robots!

Please go to the following:

https://tinyurl.com/python-robots

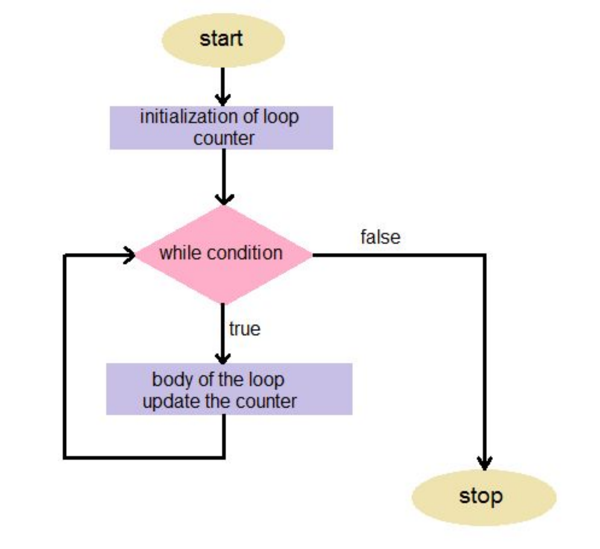
Loops and Iteration!

Questions we will answer today:

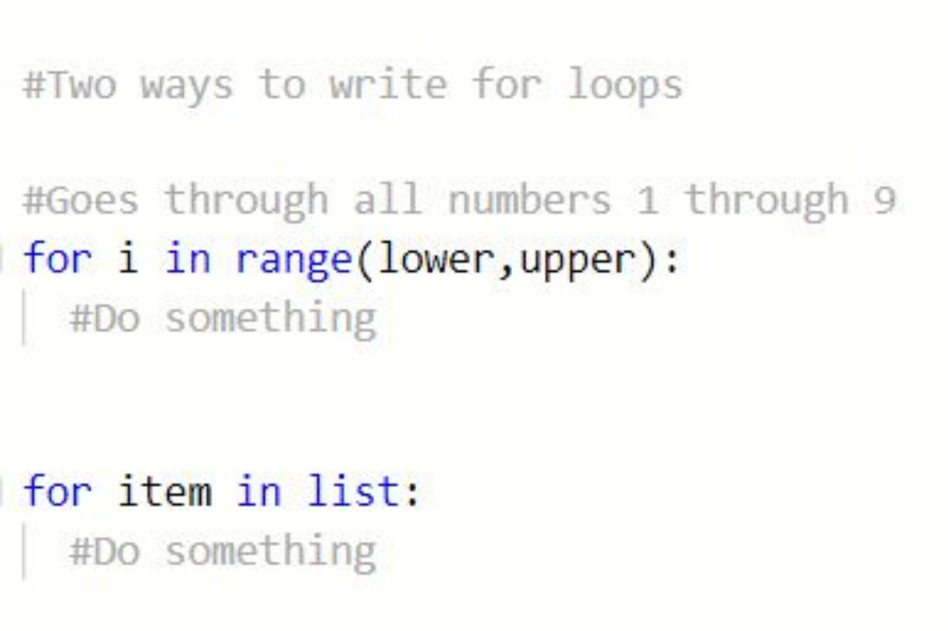
- What is iteration?
- How do we tell the computer how long we want our loops to be?
- How do we store information while we are looping?
- How can we avoid infinite loops?

What is Iteration?

- Using repetition to execute code many times.
- Types of loops:
  - for
  - while



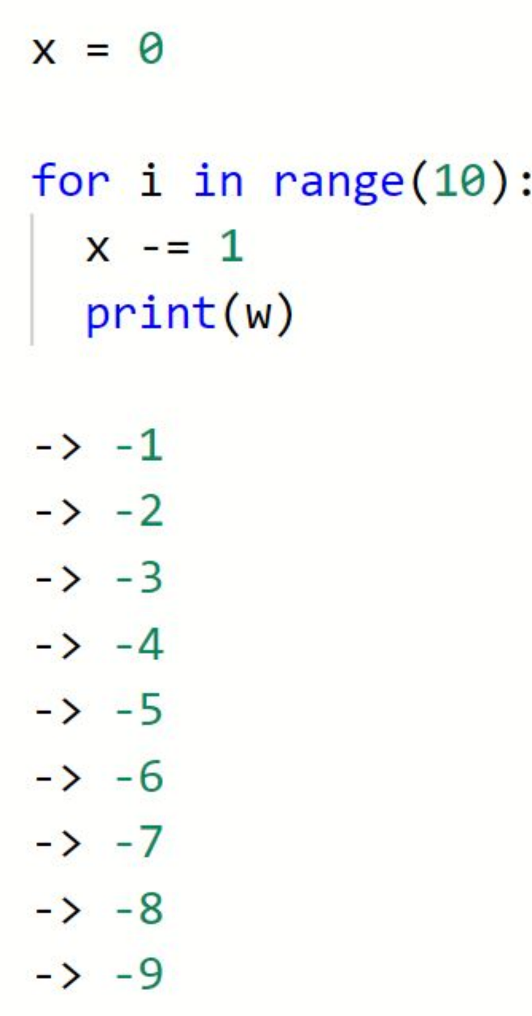
Range Function for for loops



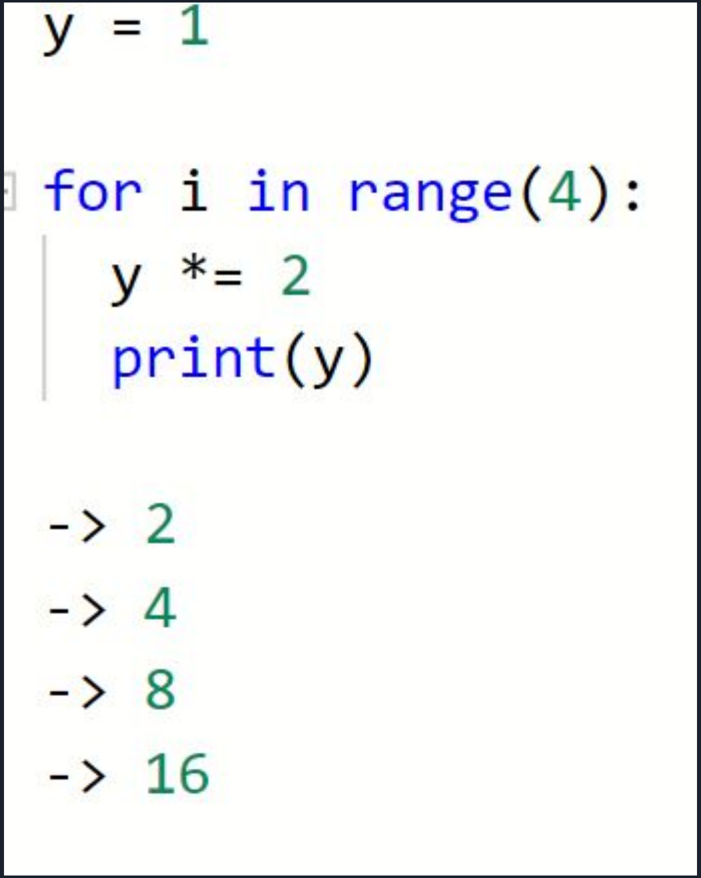
```
#Two ways to write for loops

#Goes through all numbers 1 through 9
for i in range(lower,upper):
    #Do something


for item in list:
    #Do something
```

More Math Operations!

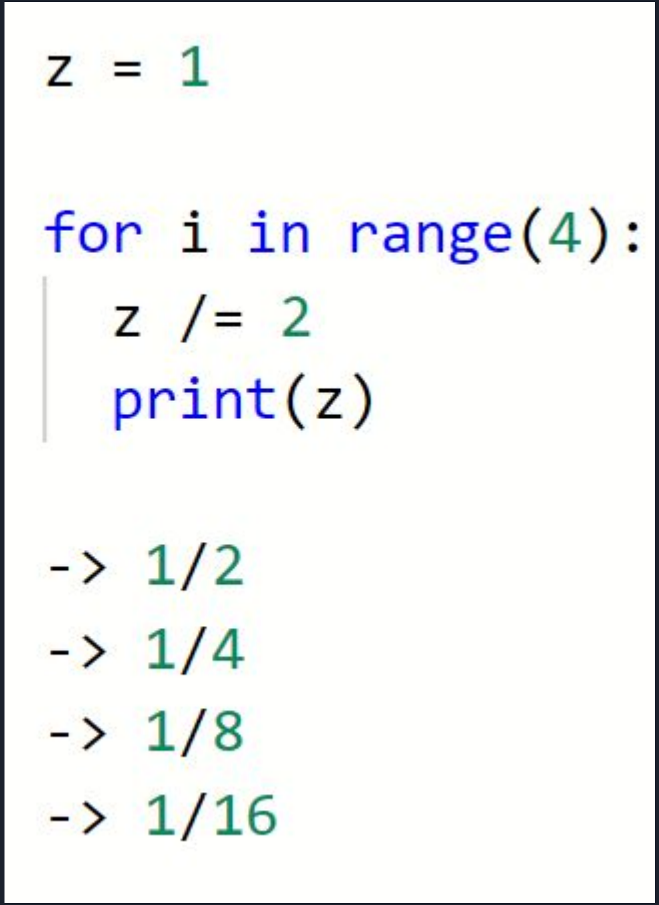
- With for loops, we can use:
  - +=: x += 1 ⟷ x = x + 1
  - -=: x -= 1 ⟷ x = x - 1
  - *=: x *= 2 ⟷ x = x * 2
  - /=: x /= 2 ⟷ x = x / 2



```
x = 0

for i in range(10):
  x -= 1
  print(w)

-> -1
-> -2
-> -3
-> -4
-> -5
-> -6
-> -7
-> -8
-> -9
```



```
y = 1

for i in range(4):
  y *= 2
  print(y)

-> 2
-> 4
-> 8
-> 16
```

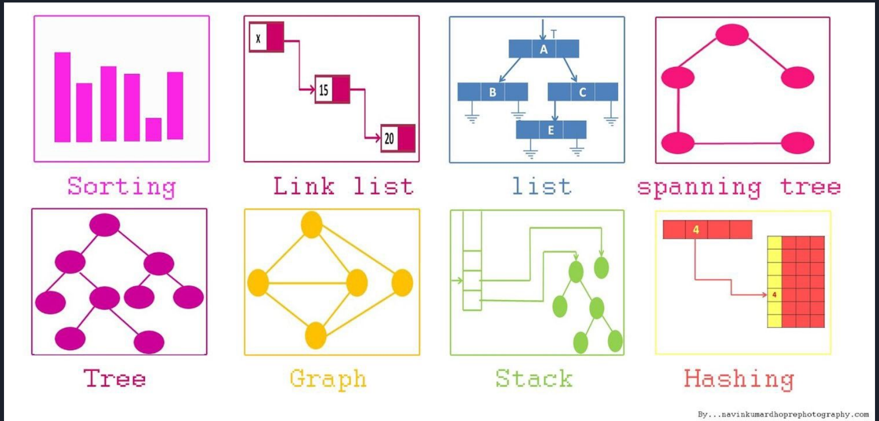


```
z = 1

for i in range(4):
  z /= 2
  print(z)
```

```
-> 1/2
-> 1/4
-> 1/8
-> 1/16
```

Data Structures: Lists and Tuples!

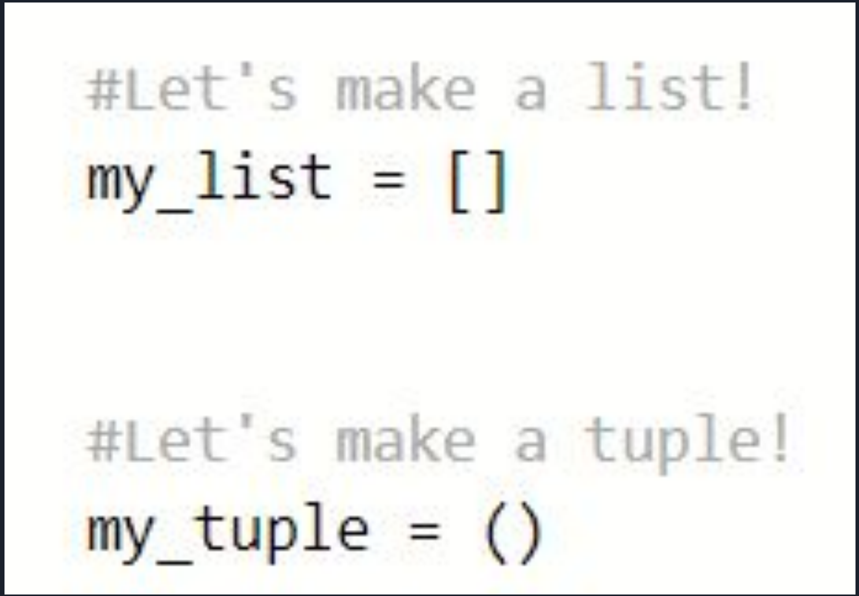
- **Data structures** store important information in Python.
- We can assign variables to be data structures too!



Lists and Tuples

- Lists and tuples store information using an **index**.
- This **index** lets us access different **elements** of our list or tuple.
- Lists and tuples store **sequences** of information.

Lists and Tuples in Python

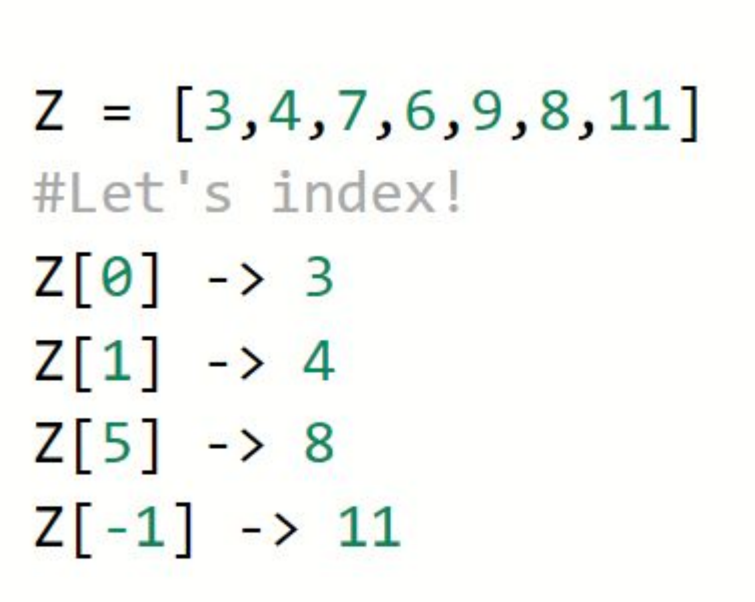


```
#Let's make a list!
my_list = []


#Let's make a tuple!
my_tuple = ()
```

Indexing in Python

- **VERY IMPORTANT:** Indexing in Python begins at 0!

- Index gives an element’s position!

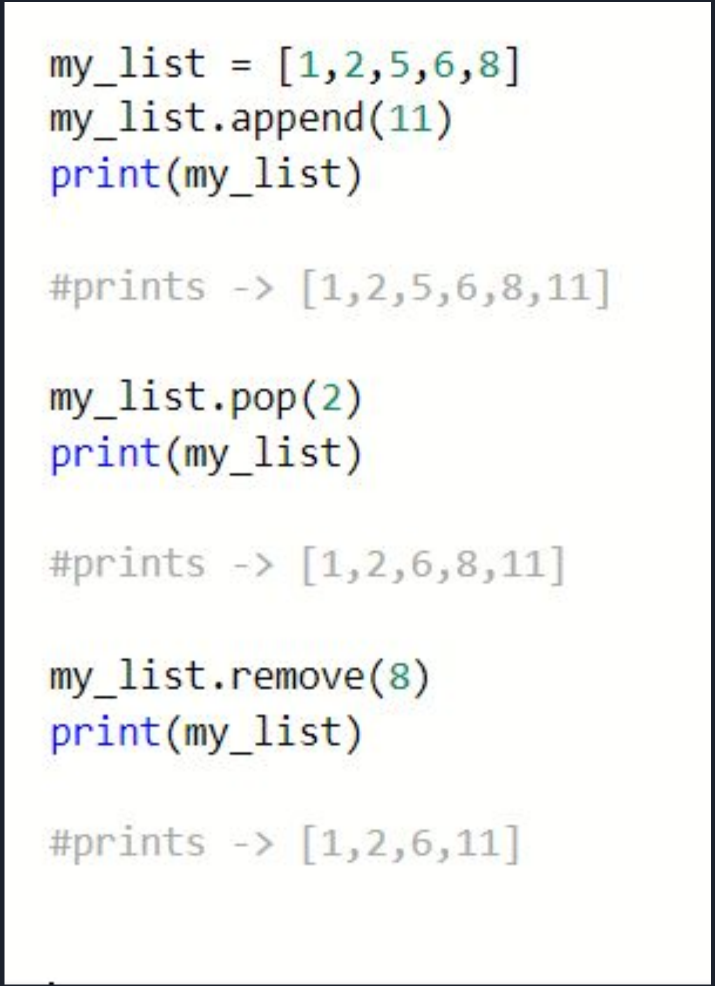


```
Z = [3,4,7,6,9,8,11]
#Let's index!
Z[0] -> 3
Z[1] -> 4
Z[5] -> 8
Z[-1] -> 11
```

Operations

- Common operations:
  - list.append(): adds an element to end of list.
  - list.pop(i): removes the element at the i position from list and returns it.
  - list.remove(x): removes the first element in list whose value equals x.
- Tuples are just like lists, except they cannot be modified.

Example Operations

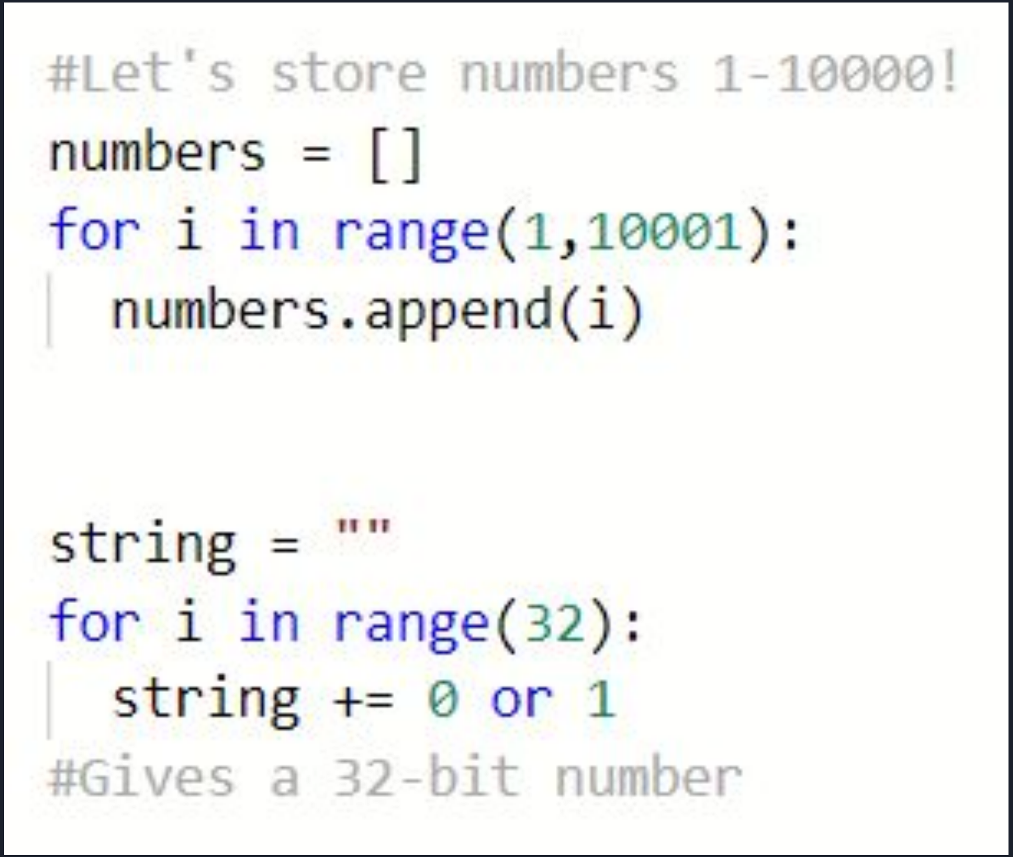


```
my_list = [1,2,5,6,8]
my_list.append(11)
print(my_list)

#prints -> [1,2,5,6,8,11]

my_list.pop(2)
print(my_list)

#prints -> [1,2,6,8,11]

my_list.remove(8)
print(my_list)

#prints -> [1,2,6,11]
```

Looping and Sequences

- Data structures we can use with **for** and **while** loops:
    - **Lists**: Add items to a list in index order.
    - **Strings**: Loop over characters in a string.

Example



```
#Let's store numbers 1-10000!
numbers = []
for i in range(1,10001):
  numbers.append(i)


string = ""
for i in range(32):
  string += 0 or 1
#Gives a 32-bit number
```

Your Turn!

Please go to the following:

https://tinyurl.com/python-loops

Warmup: Practice with Loops!

We'll go through these together! Go to:

https://tinyurl.com/loop-game

Activity: Cipher!

Let’s practice dictionaries using ciphers!

Please go to:

https://tinyurl.com/python-secret-cipher

Warmup: Practice with Loops!

- If you’re finished, try writing a short program to find the sum of ODD numbers (1,3,5,7,...,99) from 1 to 100! (Hint, use i % 2 to check if a number is ODD).
- We need:
    - Indents and “:” for **for** loops and **if** statements

Loop Warmup: Blast Off!

- Please go here: https://tinyurl.com/python-rocket
- After you run the code, let’s go through it as a class!
- ASCII Art

Review 2: Practice with Loops and Lists!

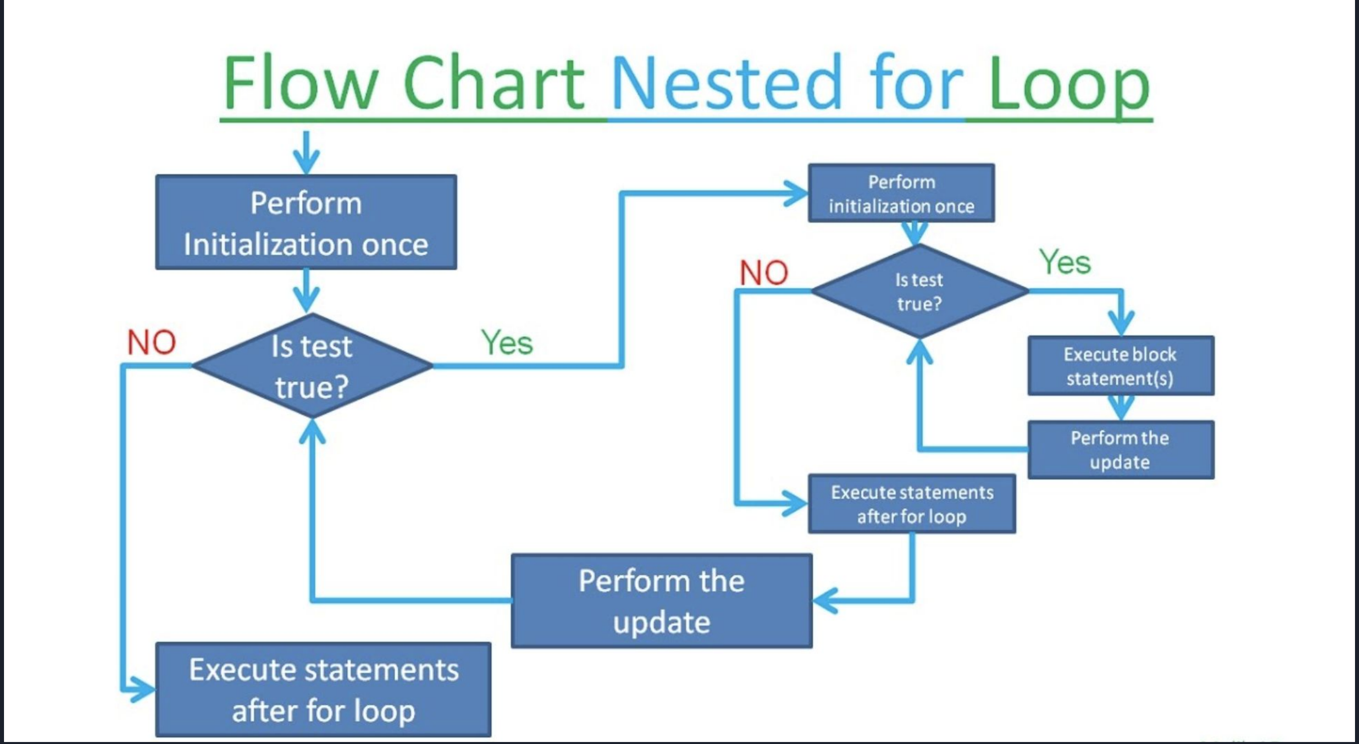
- First, let's make a list of random numbers from a distribution.
- Then, let's add these numbers to a list, and find the average of this list.
- Please go here: https://tinyurl.com/python-LLN

Law of Large Numbers (LLN)

- As we take more samples, the measured mean approaches the distribution mean!

- Very important in probability and statistics!

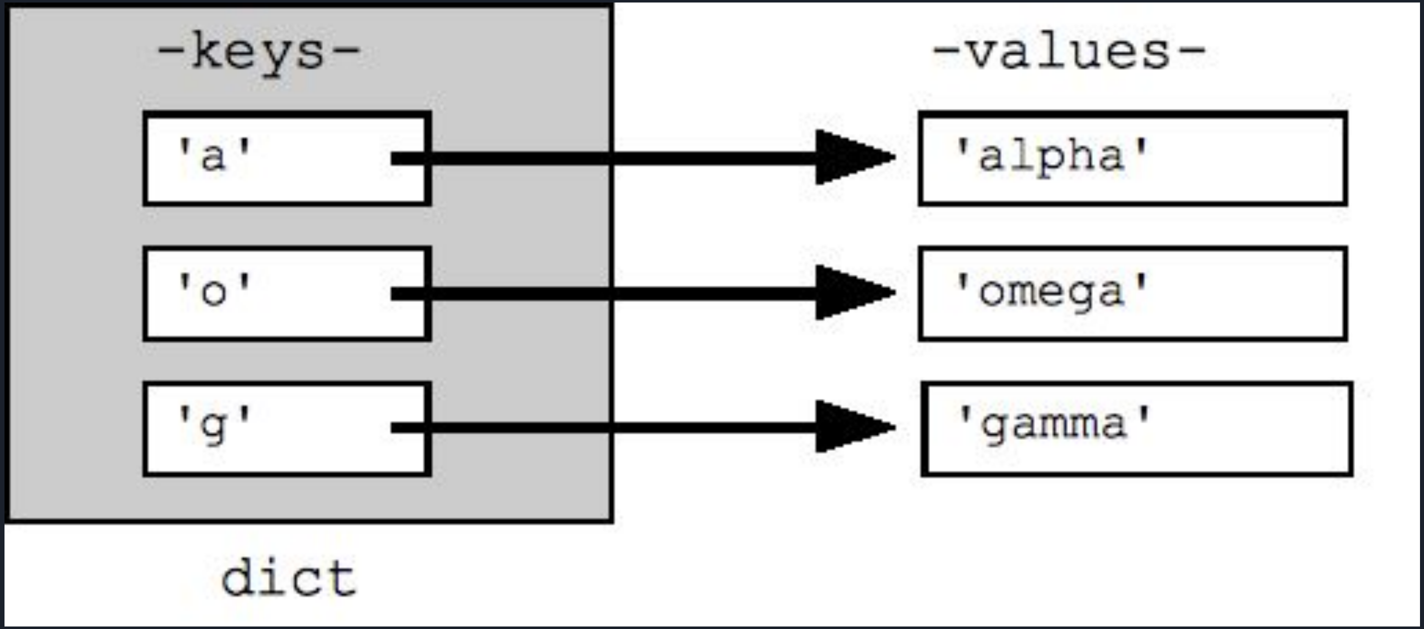
More On Nested For Loops


Flow Chart Nested for Loop
Perform Initialization once
Is test true?
NO
Yes
Perform initialization once
Is test true?
NO
Yes
Execute block statement(s)
Perform the update
Execute statements after for loop
Perform the update
Execute statements after for loop

What are Dictionaries?

- Dictionaries access information using a **key.**
- Each entry in a dictionary uses a **key-value pair**.
- Dictionaries use something known as a **hash table**.

What Are Dictionaries?

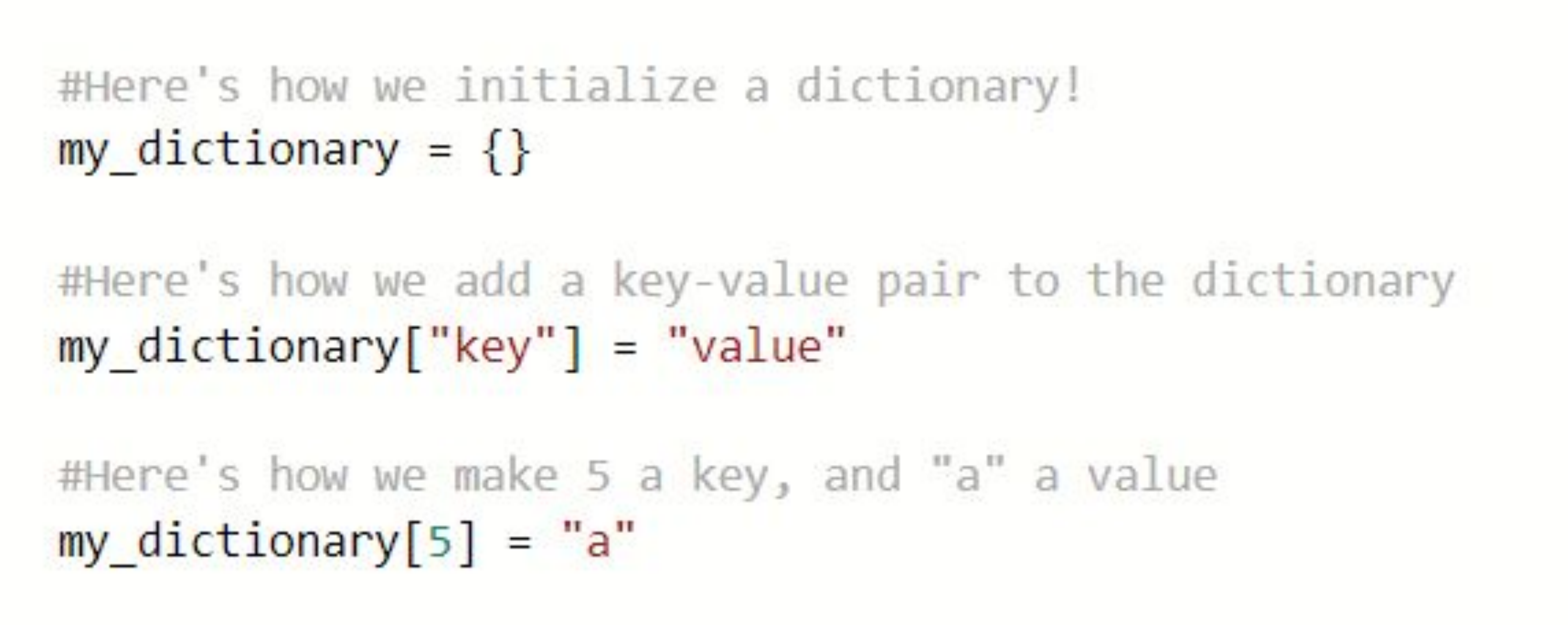


Why Use Dictionaries When We Have Lists?

- Can access information faster!
  - Important for making algorithms more efficient:
    - Faster Internet
    - Smarter robots
    - Safer vehicles
- Useful for when we can NOT order data in a logical way!

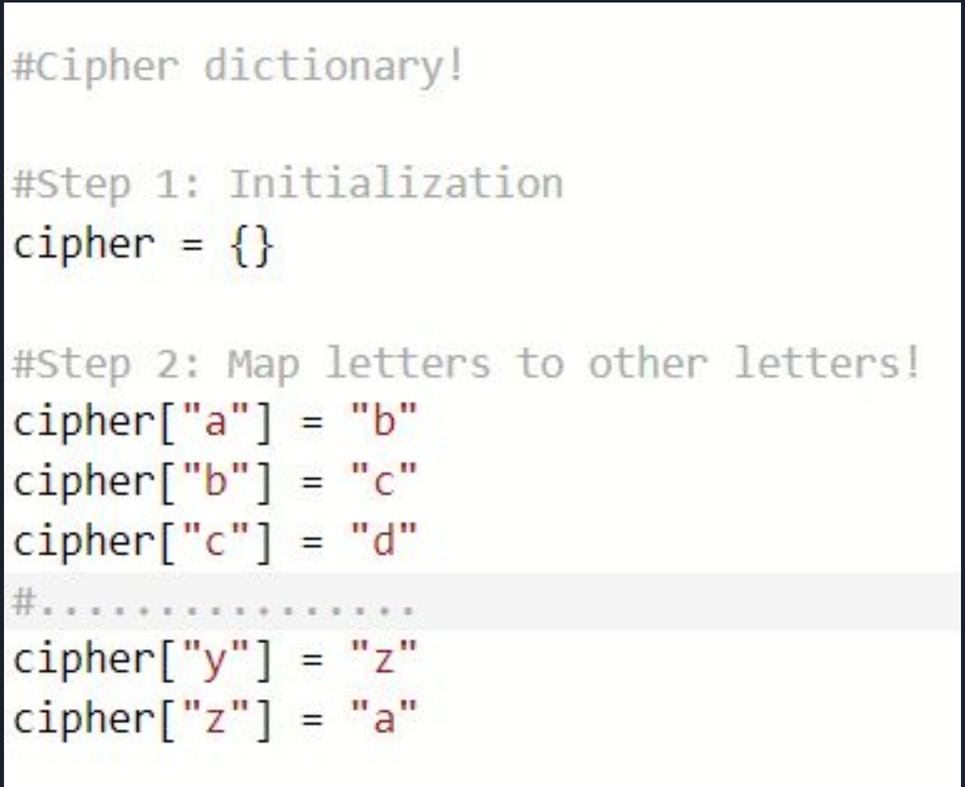
Dictionaries vs. Lists

Dictionaries in Python!



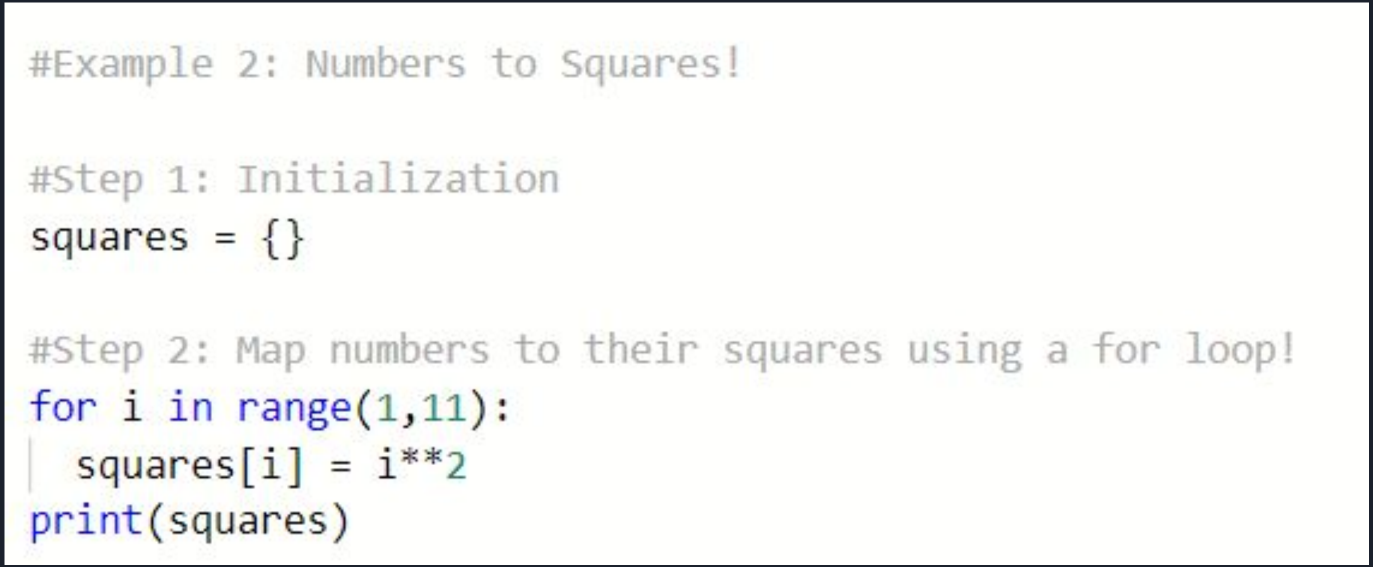
```
#Here's how we initialize a dictionary!
my_dictionary = {}

#Here's how we add a key-value pair to the dictionary
my_dictionary["key"] = "value"

#Here's how we make 5 a key, and "a" a value
my_dictionary[5] = "a"
```

Example: Cipher!



```
#Cipher dictionary!

#Step 1: Initialization
cipher = {}

#Step 2: Map letters to other letters!
cipher["a"] = "b"
cipher["b"] = "c"
cipher["c"] = "d"
#...............
cipher["y"] = "z"
cipher["z"] = "a"
```

Example: Squares!



```
#Example 2: Numbers to Squares!

#Step 1: Initialization
squares = {}

#Step 2: Map numbers to their squares using a for loop!
for i in range(1,11):
  squares[i] = i**2
print(squares)
```

What’s In a Dictionary?

For cipher example:

cipher = {“a”: ”b”, “b”: “c”,...,”y”: “z”, “z”: “a”}

For squares example:

squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}

Your Turn: Binary Number Inverter!

- Let’s turn 1’s into 0’s and 0’s into 1’s!
- Please go here: https://tinyurl.com/python-binary

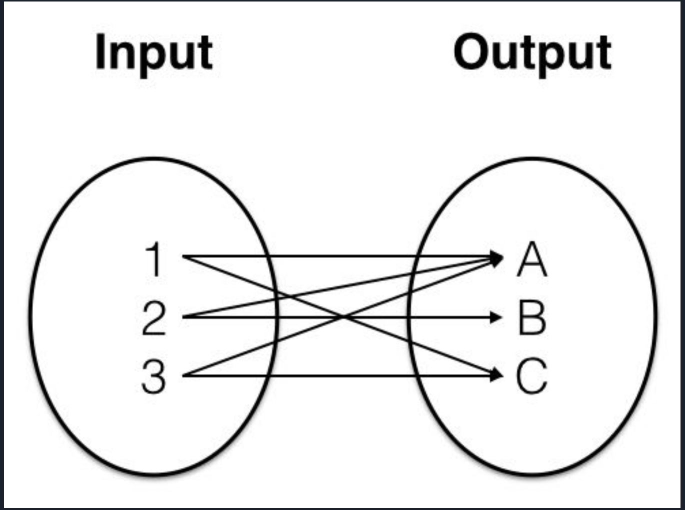
Functions!

Questions we will answer today:

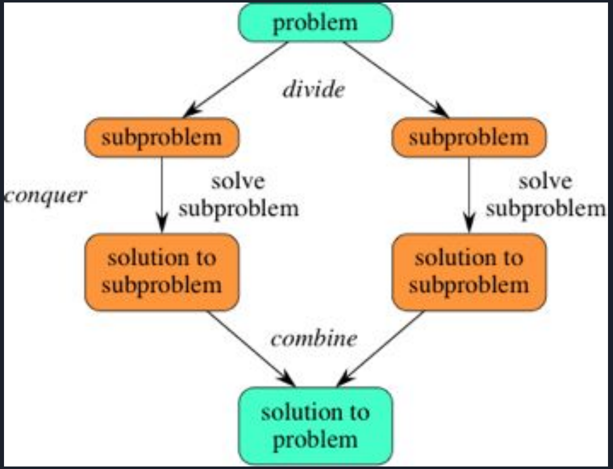
- What are functions?
- How are functions useful?
- How do we create and call functions?

What are Functions?

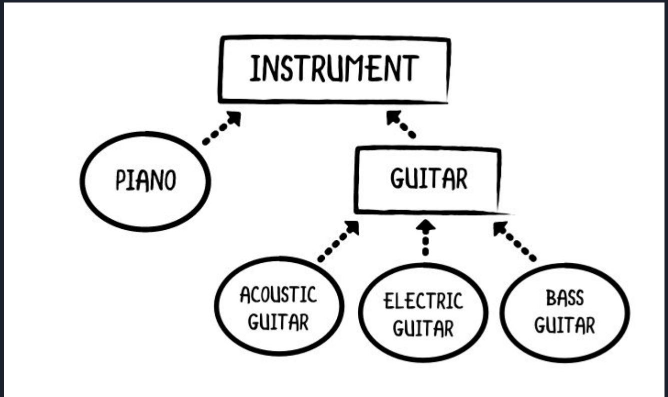
- Reusable blocks of code - no need to repeat old code!
- Input/Output relationship!



Why Are Functions Useful?



- Allows developers to share and reuse code!
  - Saves development time
  - Divide and conquer
  - Leads to object-oriented programming

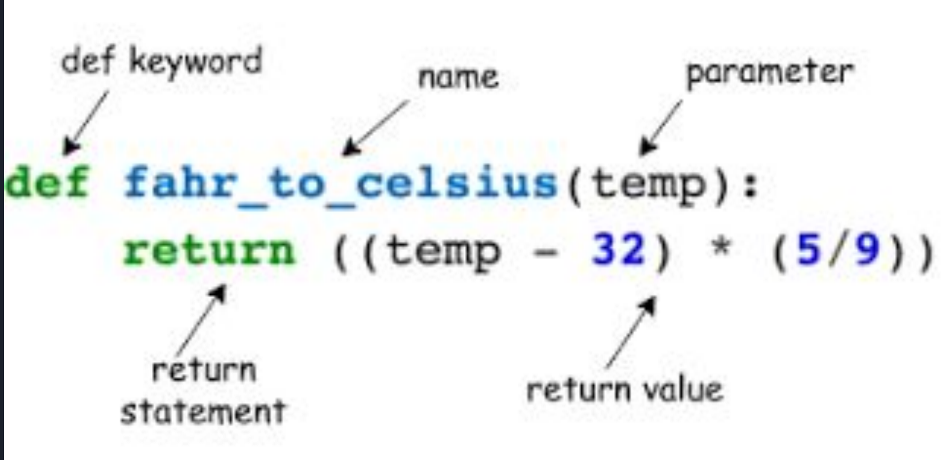


```
require File.expand_path("../../config/environment", [illegible]
# Prevent database truncation [illegible]
abort("The Rails environment [illegible]
require 'spec_helper'
require 'rspec/rails'

require 'capybara/rspec'
require 'capybara/rails'

Capybara.javascript_driver = [illegible]
Category.delete_all; Category [illegible]
Shoulda::Matchers.configure do |config|
  config.integrate do |with|
    with.test_framework :rspec
    with.library :rails
  end
end

# Add additional requires below this line. [illegible]

# Requires supporting ruby files [illegible]
# spec/support/ and its subdirectories. [illegible]
# run as spec files by default. [illegible]
# in _spec.rb will both be required [illegible]
# run twice. It is recommended [illegible]
# end with _spec.rb. You can configure [illegible]
# option on the [illegible]
```

No results found for 'mongoid'

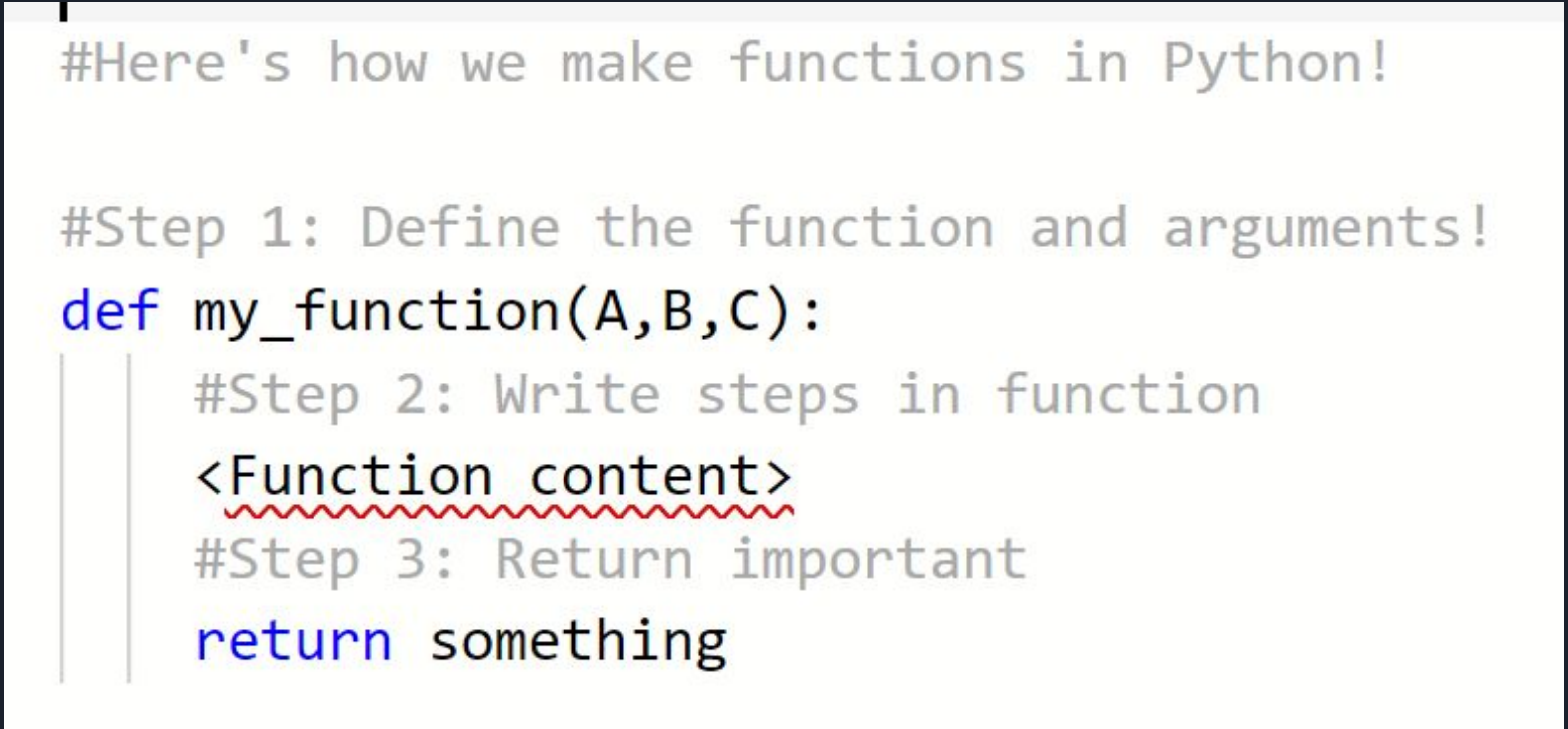
Functions in Python

- Make sure to include the def, colon, and return components of the function:



```
def fahr_to_celsius(temp):
    return ((temp - 32) * (5/9))
```

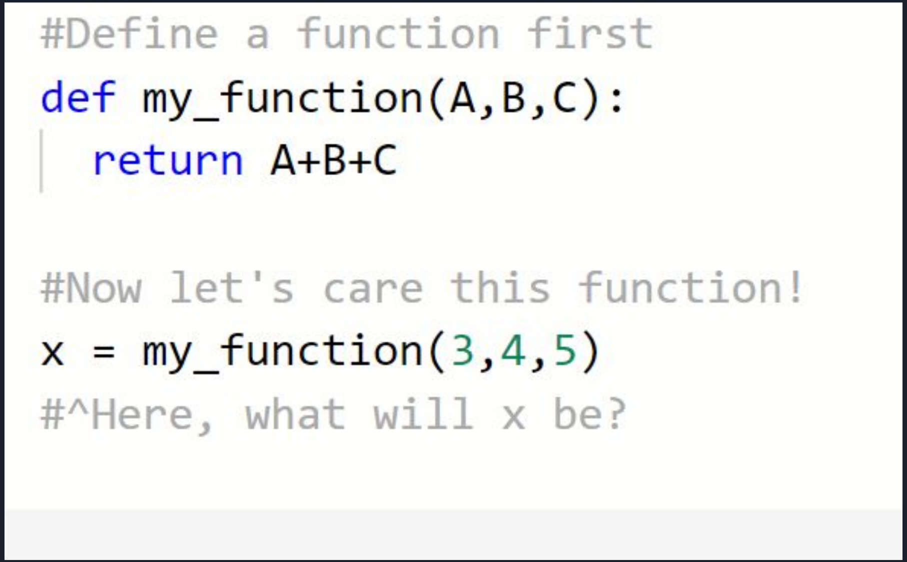
Functions in Python



```
#Here's how we make functions in Python!

#Step 1: Define the function and arguments!
def my_function(A,B,C):
    #Step 2: Write steps in function
    <Function content>
    #Step 3: Return important
    return something
```

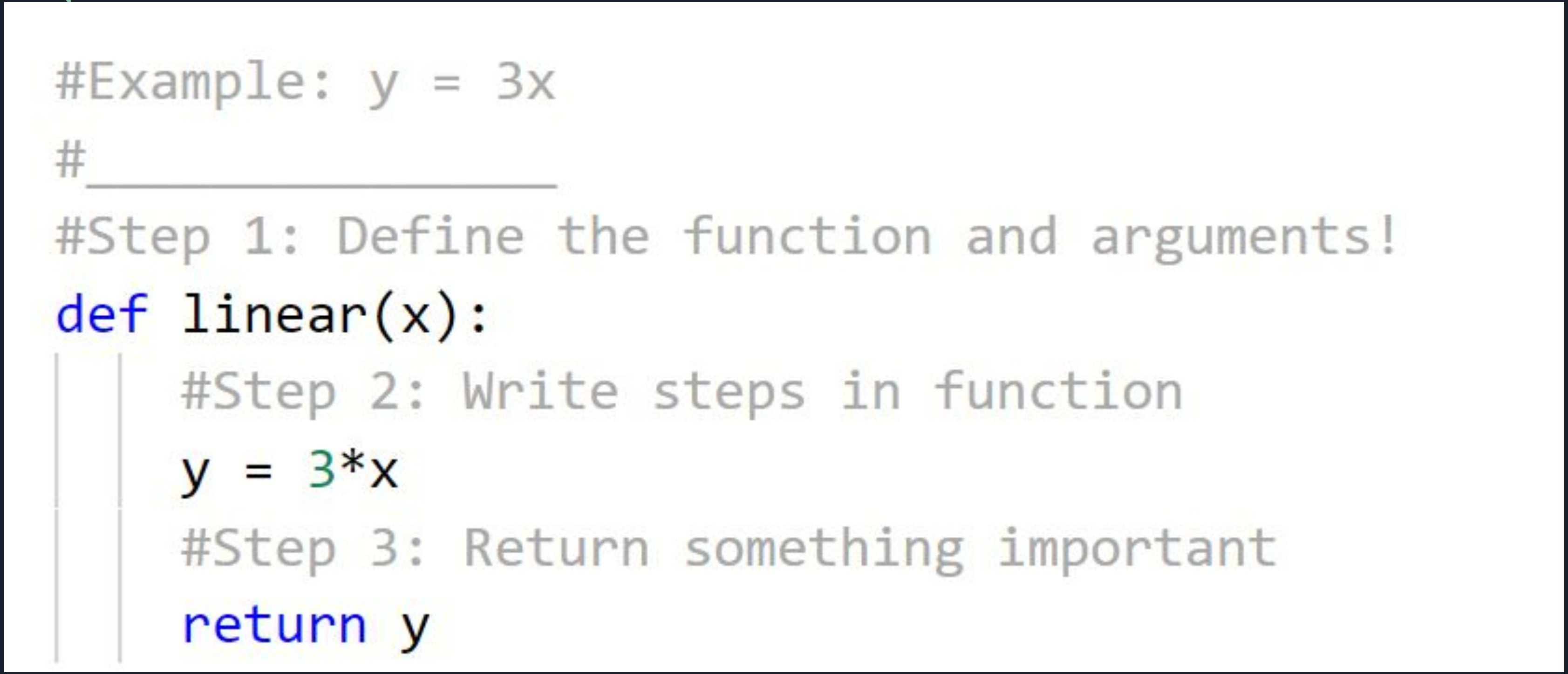
How Do We Call Functions?

- When we are ready to use a function we have defined, we can call it by:



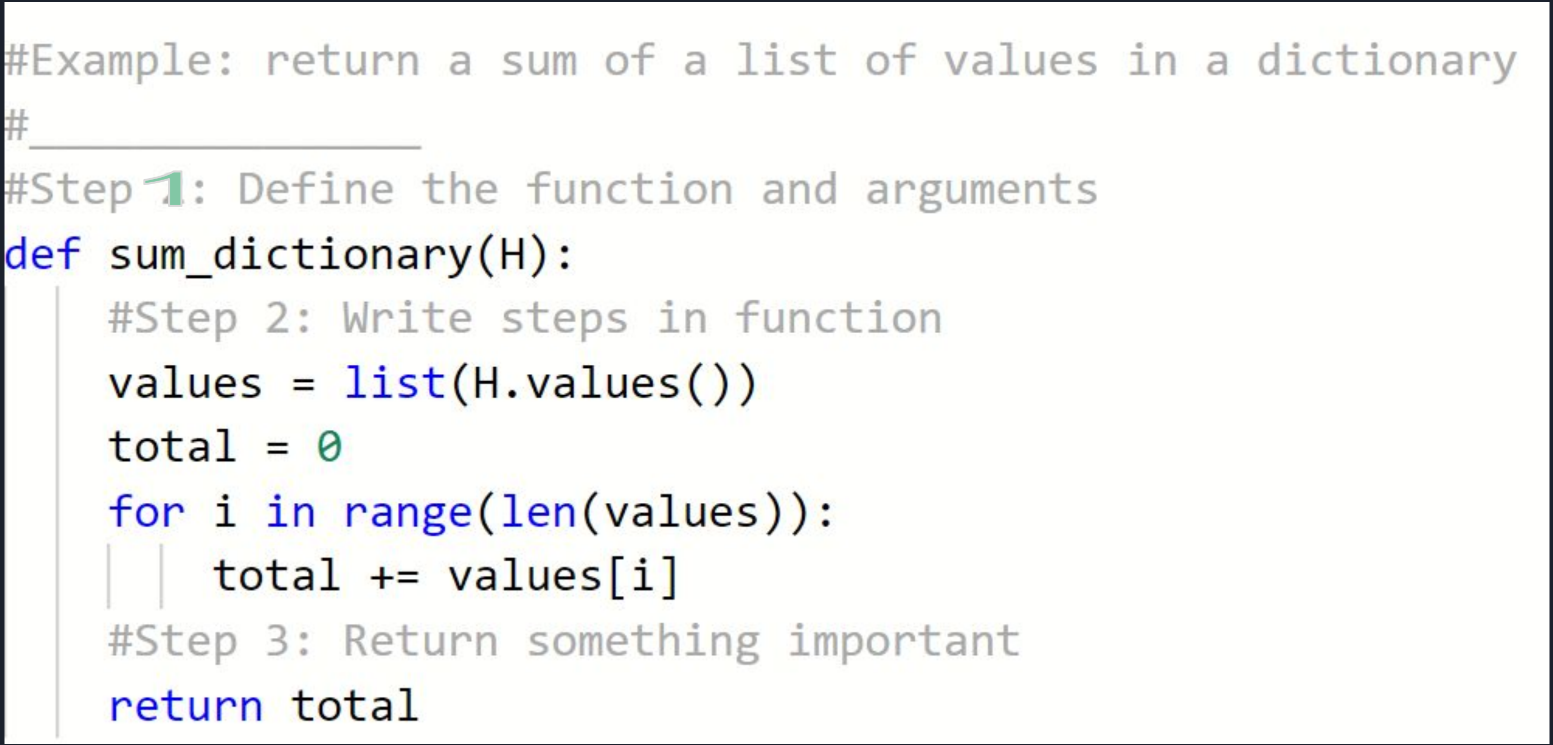
```
#Define a function first
def my_function(A,B,C):
  return A+B+C

#Now let's care this function!
x = my_function(3,4,5)
#^Here, what will x be?
```

Example: y = 3x



```python
#Example: y = 3x
#_______________
#Step 1: Define the function and arguments!
def linear(x):
    #Step 2: Write steps in function
    y = 3*x
    #Step 3: Return something important
    return y
```

Example: Sum Dictionary Values



```
#Example: return a sum of a list of values in a dictionary
#_______________
#Step 1: Define the function and arguments
def sum_dictionary(H):
    #Step 2: Write steps in function
    values = list(H.values())
    total = 0
    for i in range(len(values)):
        total += values[i]
    #Step 3: Return something important
    return total
```

Let’s Practice!

Please Google repl.it Python 3, and we’ll write some functions together!

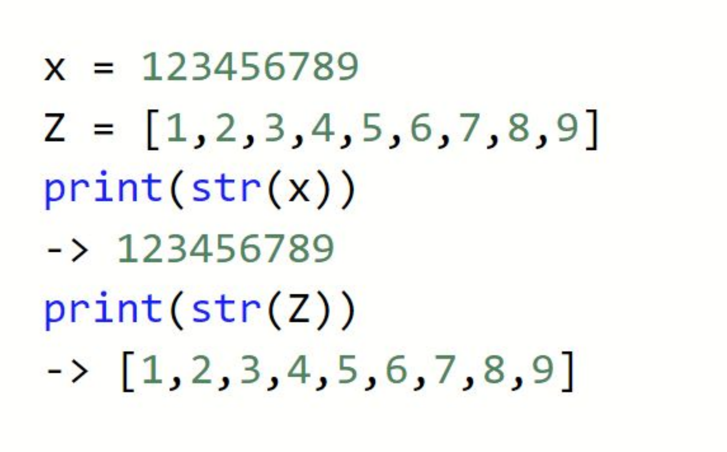
More Function Practice! Pizza and Encryption

- Please go to:

tinyurl.com/pizza-RSA4

More Ops. on Lists, Strings, and Dictionaries!

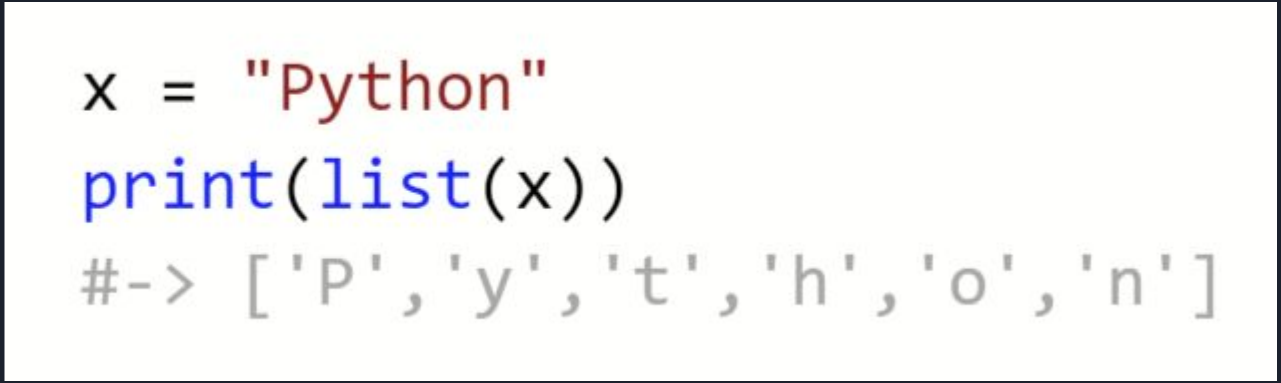
1. str(x): Converts x to a string.
    - Useful for **concatenating** (adding) strings together.



```
x = 123456789
Z = [1,2,3,4,5,6,7,8,9]
print(str(x))
-> 123456789
print(str(Z))
-> [1,2,3,4,5,6,7,8,9]
```

More List Operations!

1. list(x): Converts x to a list.
   - Useful for finding word or letter count!

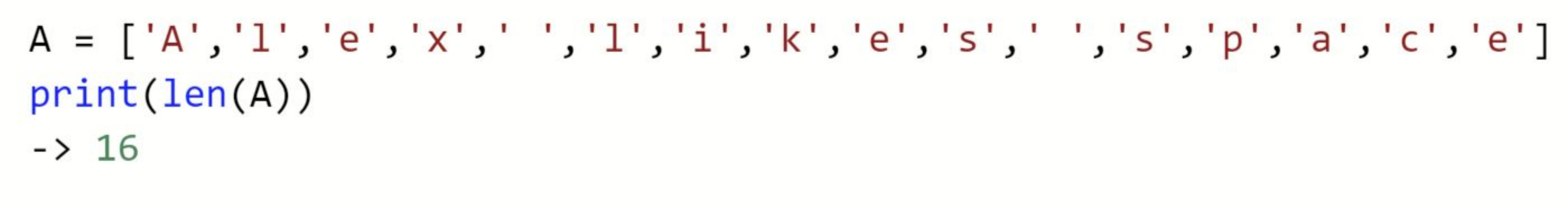


```
x = "Python"
print(list(x))
#-> ['P','y','t','h','o','n']
```

More List Operations!

2. len(A): Tells us how many elements are in list A.

- Very useful in for loops!

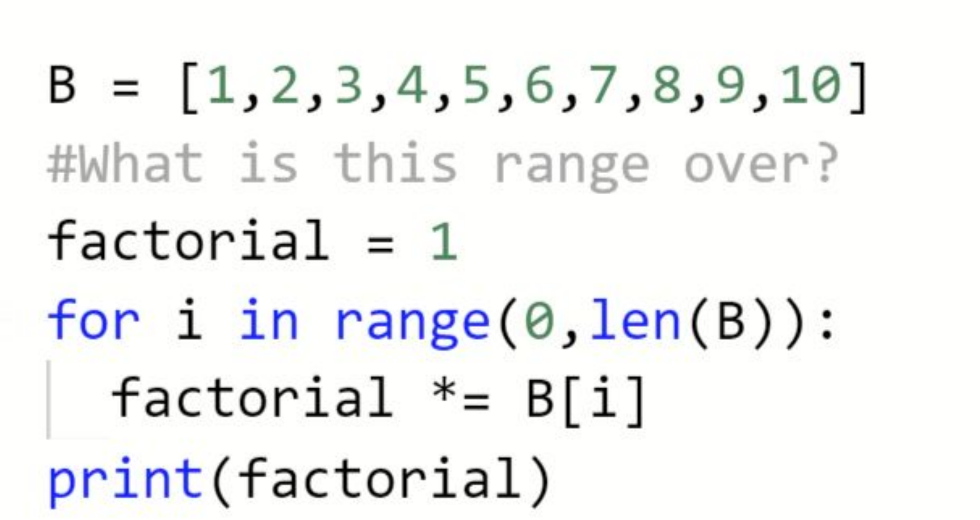


```
A = ['A','l','e','x',' ','l','i','k','e','s',' ','s','p','a','c','e']
print(len(A))
-> 16
```

More List Operations!

2. len(A): Tells us how many elements are in list A.

- Very useful in for loops!

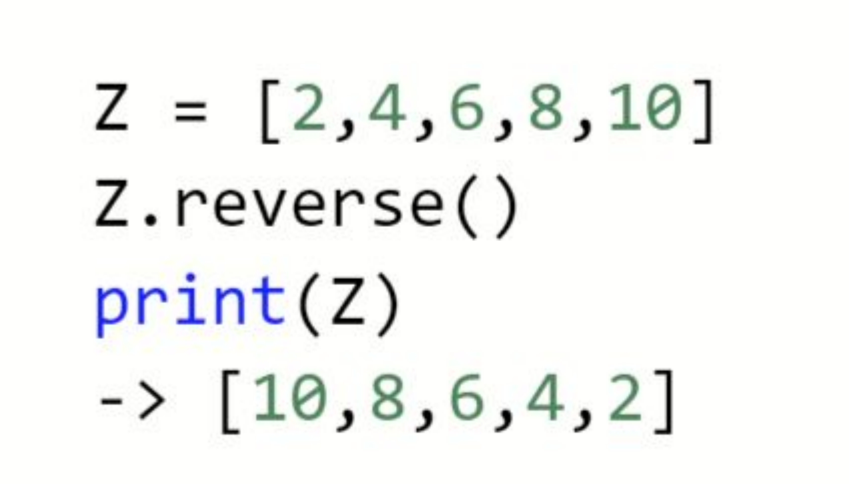


```python
B = [1,2,3,4,5,6,7,8,9,10]
#What is this range over?
factorial = 1
for i in range(0,len(B)):
    factorial *= B[i]
print(factorial)
```

More List Operations!

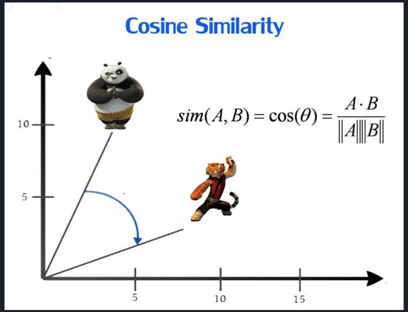
3. my_list.reverse(): Reverses order of elements in my_list.

- Useful for ciphering and encryption!



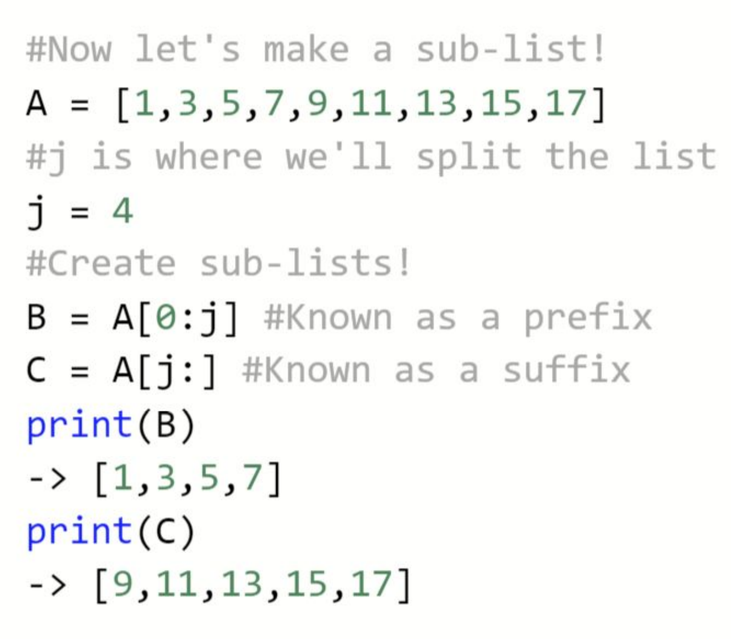
```
Z = [2,4,6,8,10]
Z.reverse()
print(Z)
-> [10,8,6,4,2]
```

More List Operations!



4. Substrings!

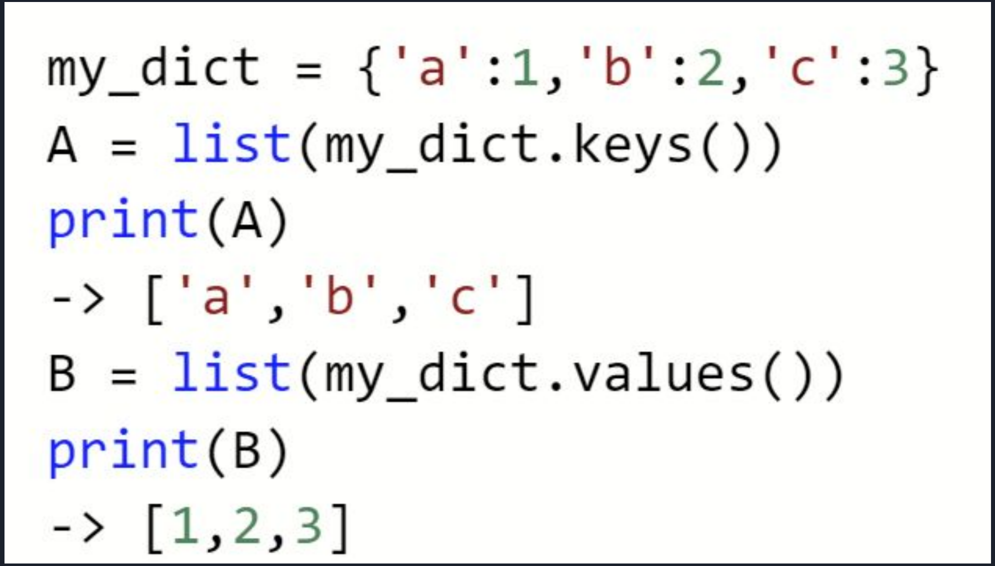
- my_list[0:j] takes first j elements of my_list (PREFIX).
- my_list[j:] takes last j elements of my_list (SUFFIX).



```
#Now let's make a sub-list!
A = [1,3,5,7,9,11,13,15,17]
#j is where we'll split the list
j = 4
#Create sub-lists!
B = A[0:j] #Known as a prefix
C = A[j:] #Known as a suffix
print(B)
-> [1,3,5,7]
print(C)
-> [9,11,13,15,17]
```

Dictionary Operations!

1. list(my_dict.keys()): Returns a list of the keys in my_dict.
2. list(my_dict.values()): Returns a list of the values in my_dict.

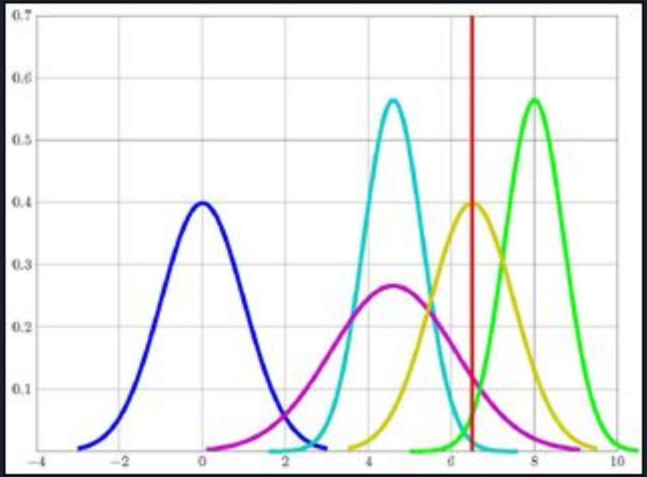


```
my_dict = {'a':1,'b':2,'c':3}
A = list(my_dict.keys())
print(A)
-> ['a','b','c']
B = list(my_dict.values())
print(B)
-> [1,2,3]
```

More Functions Practice! Probability and Stats

Please go to:

tinyurl.com/python-prob



BUGS ARE ON LINES 15, 23, 38, 39, 67, 81

CHALLENGE: Who can get the closest mean to 0.5?

- Think carefully about the number of coin flips…
- When you’re ready, go here:

  https://tinyurl.com/python-prob-chall

- On line 48, change n to the number of coin flips you want to use.

ALTERNATIVE LESSON PLANS:

Recursion, Graph Theory, Dynamic Programming, Regression

*NOTE: These are only skeleton slides.

Nested and Helper Functions

- Divide and Conquer idea
- We can define functions within functions!

<code block about nested and helper functions>

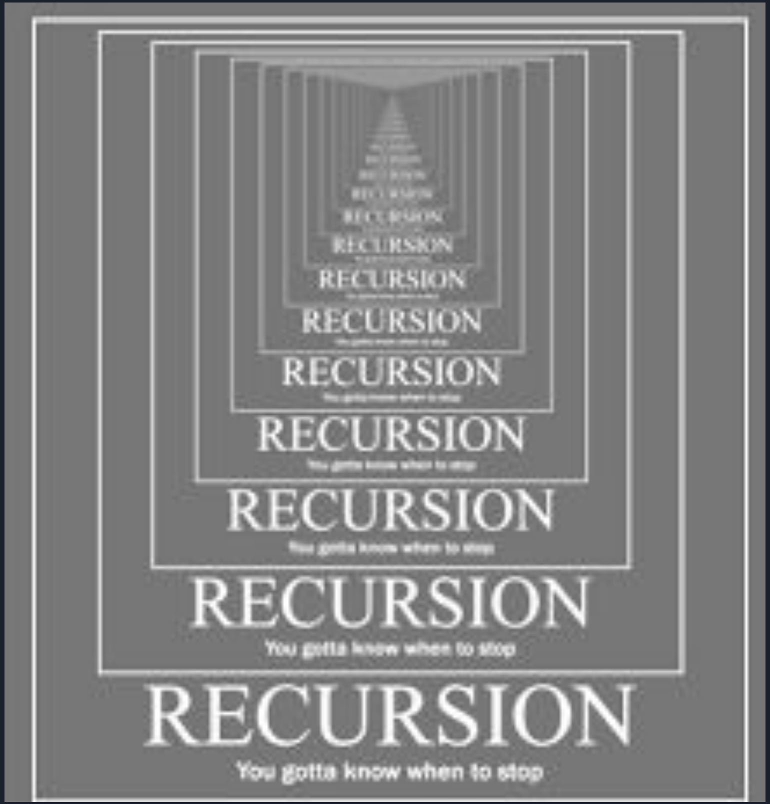
Recursion

Questions we will answer today:

- What is recursion?
- What kinds of problems can be solved using recursion?
- How do we define base cases for recursion?

What is Recursion?

- Calling a function repeatedly with smaller inputs.
  - Use a function that calls itself!
  - Uses base cases to make sure we don't call forever.



Anatomy of a Recursive Function

<code block, diagram, and definitions of different parts of a recursive function (definition, base case, inductive step, and return step)>

Your Turn!

<Recursive Problem 1>

<Recursive Problem 2>

Duality: Iteration vs. Recursion

- **Iteration** and **recursion** can be used to solve the same problems!

<Problem with iteration> <Problem with recursion>

Your Turn!

<Recursive or Iterative Exercise 1>

<Recursive or Iterative Exercise 2>

Classes and Methods!

Questions we will answer today:

- What are classes and methods?
- What is object-oriented programming?
- How do we initialize **instances** of a class?

What are Classes and Methods?

- Classes are used to create **objects**. Objects have:
  - **Attributes,** which contain object information.
  - **Methods,** which are function methods you can call on the object.

Initialization Method in Python

- We first use an initialization **method** when we write a class.
- This initialization method lets us to assign values to an object’s **attributes**.

Example Initialization Method

<code block for example initialization method>

Self in Classes

- Self is used to denote the name of the object we create when we use a class.
- Although arbitrary, is the standard for Python.

When Do We Use Self?

- We use self whenever:
  - We define a class method.
  - We assign a class attribute to a value.
  - We call a method in a class.

Example Uses of Self

<code block(s) for using self>

An Example In Python

<Example of an implementation of an object via a class in Python>

Creating an Instance of a Class

- Creating an object using a class is known as creating an **instance** of a class. Below is an example.

<code for creating an instance of a class>

Your Turn!

- Help me complete the initialization step!
  - <Attribute 1>
  - <Attribute 2>
  - <Attribute 3>
- Now, define this method!
  - <initialization method for example class to implement>

(If We Have Time) Your Turn!

- Let’s do it again! Implement a method that <add additional method functionality here>.

<code block for method to be added>

Graph Theory!

Questions we will answer today:

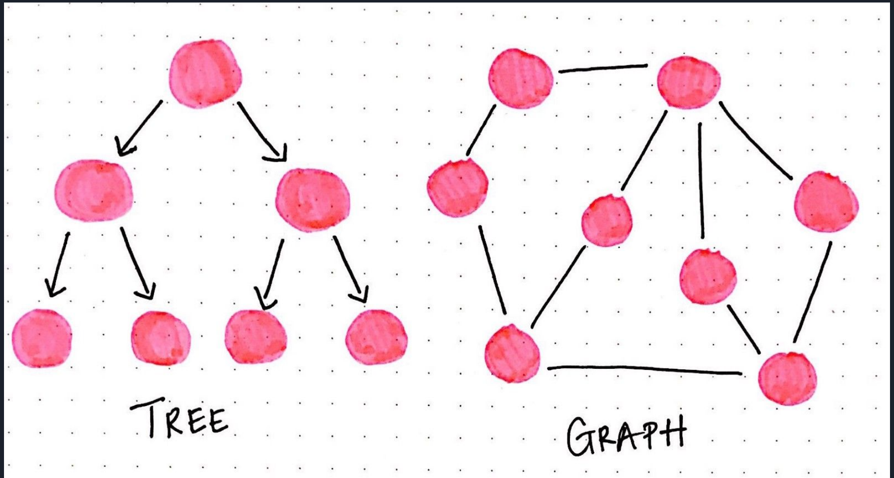
- What is a graph?
- How do we create graphs in Python?
- How do we find the shortest path in a graph?

What Are Graphs?

- Graphs are used to model pairwise relations between objects.
- Made up of:
  - **vertices**, **nodes**, or **points**

    AND

  - **edges**, **arcs**, or **lines.**

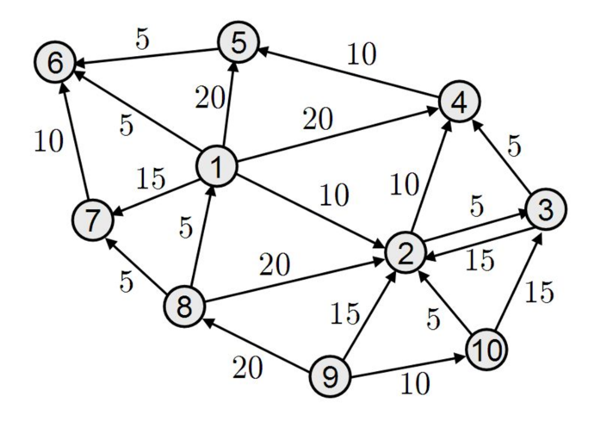


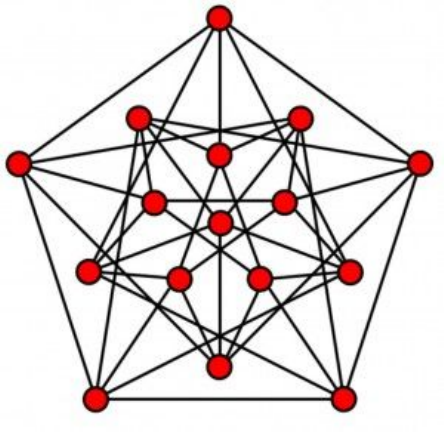
How do We Create Graphs In Python?

- Can create graphs using:
  - ***Adjacency Lists***: <Python code for adjacency list>
  - **Adjacency Matrix**: <Python code for adjacency matrix>
  - **Objects**: <Python code for Node and Edge Objects>

Shortest Paths

- Solving Shortest Path Problem:
  - **Breadth-First Search (BFS),** if edges **unweighted**.
  - **Dijkstra's Algorithm,** if edges **weighted**.





Shortest Paths 1: Breadth-First Search!

- Breadth first search finds shortest paths by finding **level sets** in a graph.
- Returns the **shortest path** after we explore the entire graph!
- Let’s try it out!

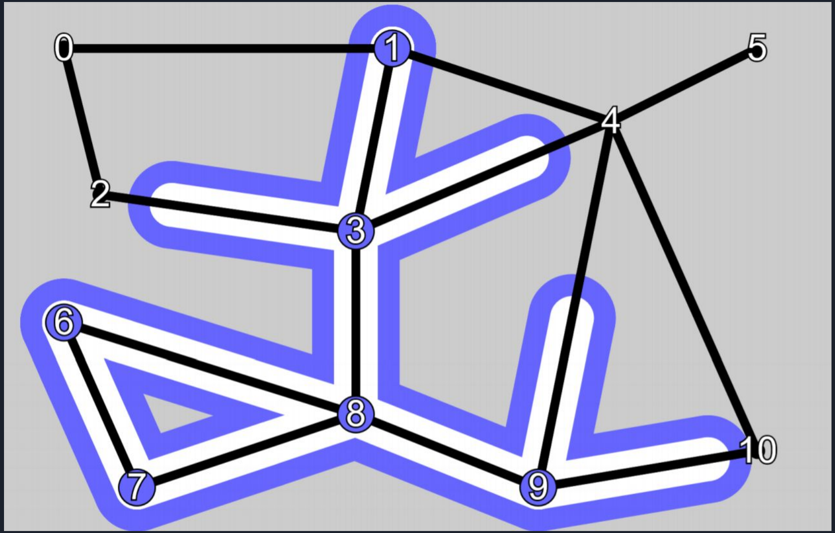
Your Turn! Breadth-First Search

- Try running BFS on your computer! After the end of the BFS function, print d[<some vertex in the graph>]. This will print the shortest path from <the name of the source vertex> to <the name of the vertex>.

Shortest Paths 2: Dijkstra’s Algorithm

- Useful if we have edge weights not equal to 1!

- “Expanding Frontier” finds shortest path.



Your Turn! Dijkstra’s Algorithm

- Try running Dijkstra’s Algorithm on your computer! After the end of the Dijkstra function, print d[<some vertex in the graph>]. This will print the shortest path from <the name of the source vertex> to <the name of the vertex>.

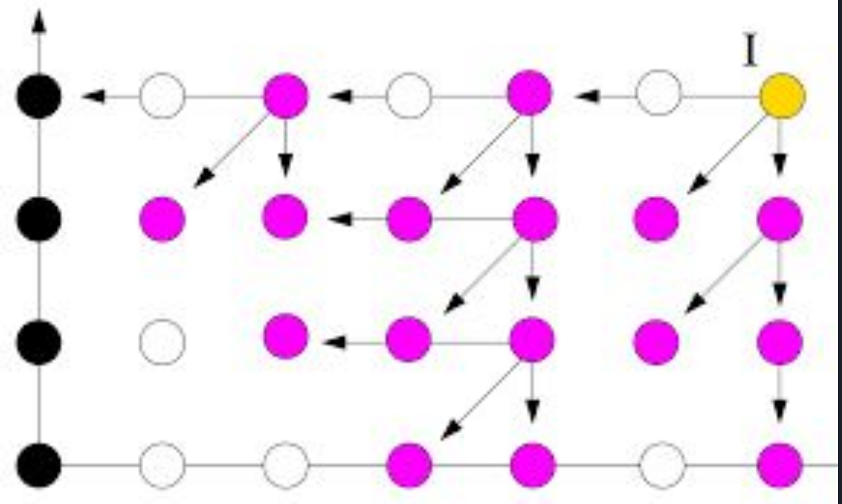
Dynamic Programming!

Questions we will answer today:

- What is dynamic programming?
- How do we solve problems with dynamic programming?
- How do we implement dynamic programming problems in Python?

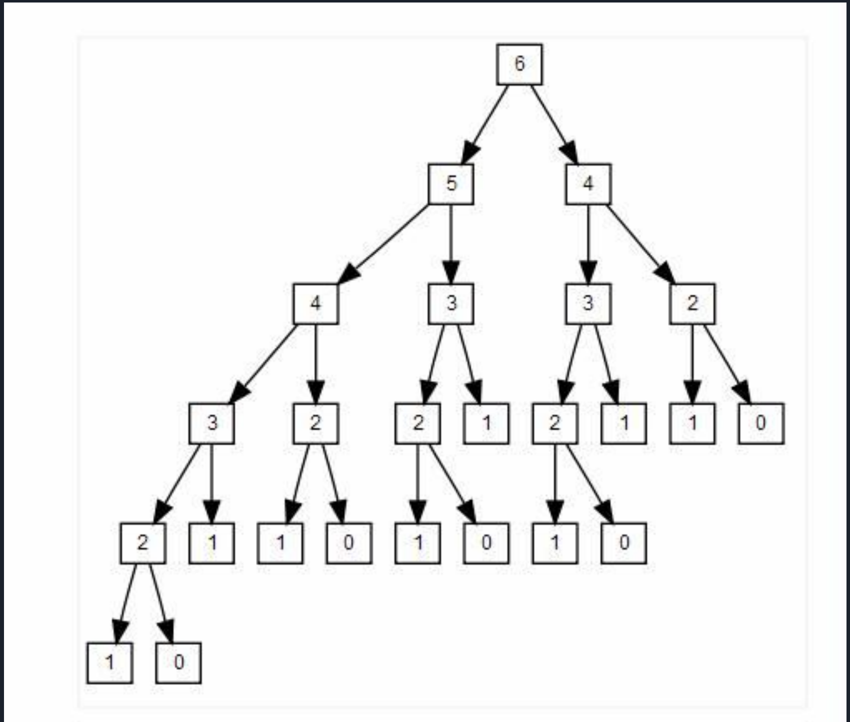
What Is Dynamic Programming?

- Dynamic programming ~“smart brute force”.
- Uses **memoization:**
  - Store smaller solutions in a dictionary to solve bigger problems!
  - More efficient!



Top-Down vs. Bottom-Up

- Two approaches, either (usually) works!
  - **Top-down ~ recursion**!
  - **Bottom-up ~looping**!



Top-Down vs. Bottom-Up with Fibonacci Numbers

<code block for Fibonacci numbers code for top-down>

<code block for Fibonacci numbers code for bottom-up>

Practice!

- The best way to learn dynamic programming is to practice! Let’s do some here.

<text for DP Practice Problem Statement 1>

More Dynamic Programming Practice!

<text for DP Practice Problem Statement 2>

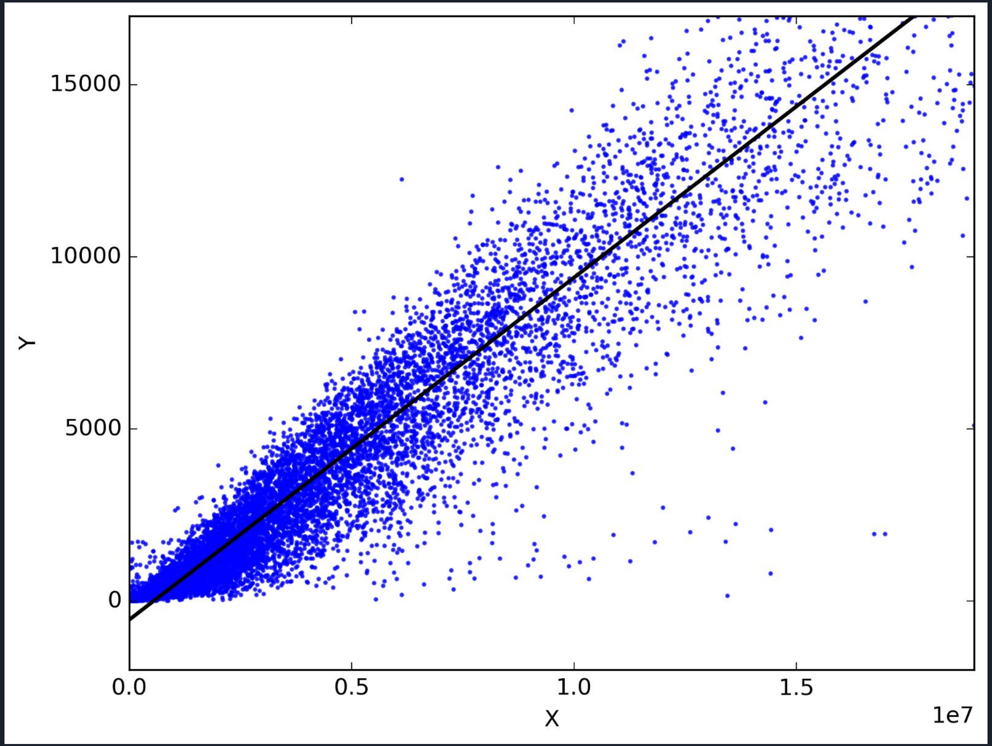
What is Regression?

- Regression is an **estimation** technique.
- Regression enables us to **predict.**
- Today, we will talk about linear regression!

Linear Regression

- In linear regression, we will find the relationship between our input values, x, and our output values, y.
- Our model for linear regression will consist of two **parameters:**
  - **A: Model slope.**
  - **B: Model offset.**
- We won’t talk about it here, but if you are interested, we will be using least squares and gradient descent methods to find these parameters.

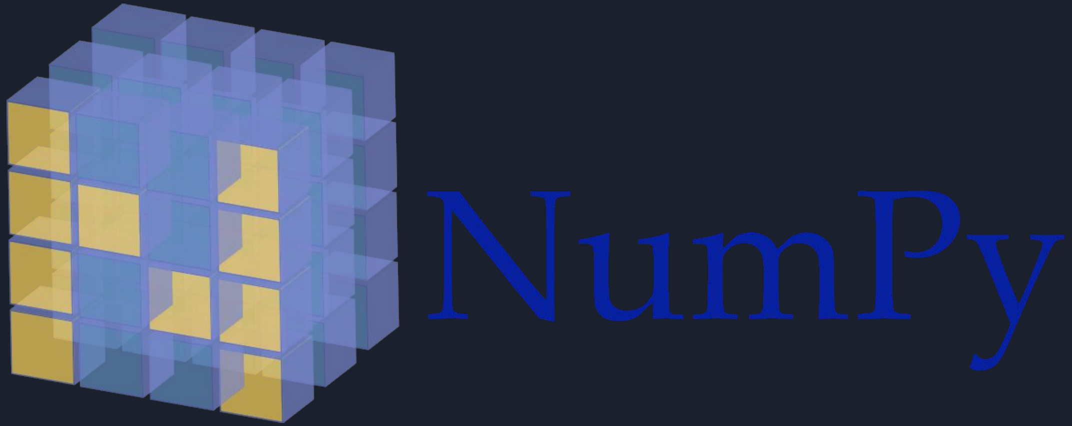
Illustrative Example



Linear Regression in Python

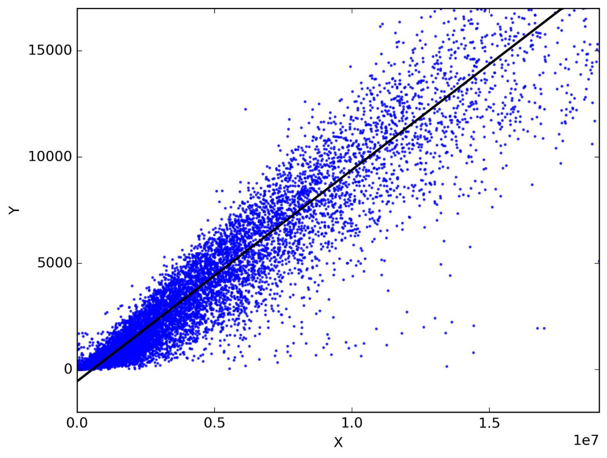
- **numpy** for **vector** and **matrix** operations!

- **np.array([[]])** data structure for **linear algebra** operations.



Your Turn Part 1: Estimating Parameters

- Using the data I have provided you, try running the code **linear_regression.py** in Repl.it and please tell me your parameters!



Your Turn Part 2: Predicting an Output

- Now, using your parameters and the input x = <pick a number>, can you tell me what the predicted output is?

Your Turn Part 3: Evaluating A Linear Fit

- We can evaluate how good our fitting of our data is by calculating the **coefficient of determination**, or $\boldsymbol{R^2}$ value.
- The closer this coefficient is to 1, the better our fit is!
- What $\boldsymbol{R^2}$ value did you get for this coefficient from the previous example?

Final Tips For Writing Good Code

- **Divide and Conquer!**
- **Use Comments!**
- **Use functions and loops to avoid repetition!**
- **Variable Names!**
- **Learn Errors!**
- **Syntax:**
  - **Indents**
  - **Parentheses (), Brackets [], Curly Braces {}**
  - **Colons :**