Python Bootcamp

Women Who Code

Session I

Language Basics

Curriculum

Unit 1 :

- Install Python
- Command Line Basics
- Running Python Code
- Git and Github Overview

Unit 2:

- Basic DataTypes
- Lists
- Dictionaries
- Tuples
- Sets
- Booleans
- File I/O (Reading data (Working with various data formats like CSV, JSON, XML, Excel and even PDFs!)

Unit 3 :

- Loop Statements
- If Ellf
- For Loops
- While Loops

- Unit 4:
- Methods
- SysArgs

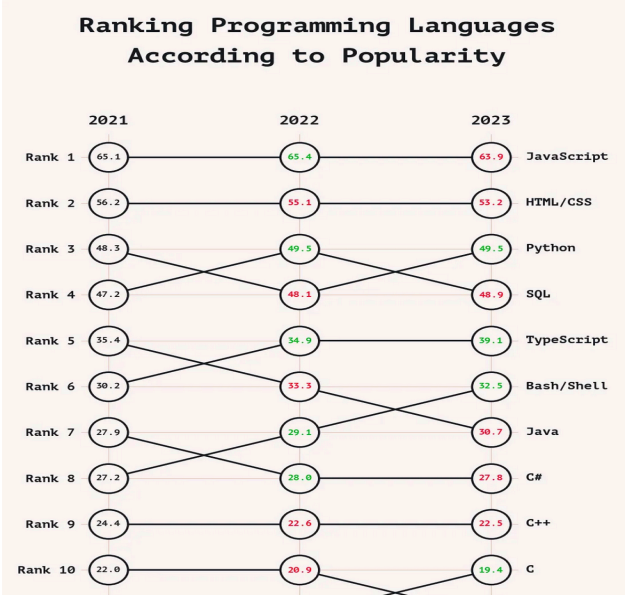
- Unit 5:
- Python Project Set up

Unit Template

- Definition
- Walk Through
- Example
- Exercise 2 Questions
- Reference

Introduction & Set-up

- Why Python –simple , interpreted, large community support
- **19 Guiding principles of Python !**
  - Beautiful is better than ugly.
  - **Explicit is better than implicit.**
  - **Simple is better than complex.**
  - **Complex is better than complicated.**
  - Flat is better than nested.
  - Sparse is better than dense.
  - **Readability counts.**
  - Special cases aren't special enough to break the rules.
  - Although practicality beats purity.
  - **Errors should never pass silently.**
  - Unless explicitly silenced.
  - In the face of ambiguity, refuse the temptation to guess.
  - There should be one-- and preferably only one --obvious way to do it.[a]
  - Although that way may not be obvious at first unless you're Dutch.
  - Now is better than never.
  - Although never is often better than *right* now.[b]
  - **If the implementation is hard to explain, it's a bad idea.**
  - If the implementation is easy to explain, it may be a good idea.
  - Namespaces are one honking great idea – let's do more of those!



https://levelup.gitconnected.com/the-fastest-growing-and-fastest-shrinking-programming-languages-644949cd1de6

Introduction & Set-up

- Python Installation : https://www.python.org/downloads/
- Repository of Packages : https://pypi.org/
- Jupyter Notebook On the Web : https://r.search.yahoo.com/_ylt=AwrFFy0ugtFkXVcrWA0PxQt.;_ylu=Y29sbwNiZjEEcG9zAzEEdnRpZAMEc2VjA3Ny/RV=2/RE=1691480751/RO=10/RU=https%3a%2f%2fjupyter.org%2f/RK=2/RS=ji4YEFmShxKDAYhrARL7AAuhcg8-

Running Python

- Install Python
- Go to your command line
- Type “python”

Program Basics - Variables in Python

- Python has **commenting** capability for the purpose of in-code documentation.
  - **Comments** start with a #, and Python will render the rest of the line as a comment:
- **Variables** are containers for storing data values.
  - Python has no command for declaring a variable.
  - If you want to specify the data type of a variable, this can be done with casting.
  - Variable names are case-sensitive.
  - **Camel Case:** exampleVariable
  - **Pascal Case : E**xampleVariable
  - **Snake Case :** example_variable
  - Python allows you to assign values to multiple variables in one line:
  - And you can assign the *same* value to multiple variables in one line:
  - If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.
  - Output : The Python print() function is often used to output variables. type() function.
  - Scope of variables:
    - Variables that are created outside of a function (as in all of the examples above) are known as global variables.
    - **Global variables** can be used by everyone, both inside of functions and outside.
    - **Local Variable :** If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.
    - If you use the global keyword, the variable belongs to the global scope:
    - To change the value of a global variable inside a function, refer to the variable by using the global keyword

```
#This is a comment
print("Hello, Pythoners!")

``` Hello,
Pythoners ```

Illegal Variable Names:
2myvar = "John"
my-var = "John"
my var = "John"


x, y, z = "Orange", "Banana", "Cherry"
x = y = z = "Orange"

fruits = ["apple", "banana", "cherry"]
x, y, z = fruits

print(x)
print(y)
print(z)
```

Exercise:

Create a variable named **myVariable** and assign the value **MYVAR** to it.
Create a variable named **myVariable** and assign the value **100** to it.
Display the sum of 25.5 + 10.4 , using two variables: **myVariable1 and myVariable2.**
Create a variable called **myVariable3**, assign **myVariable1 + myVariable2** to it, and display the result.
Remove the illegal characters in the variable name: **10my-car_brand = ”MYBRAND”**
Insert the correct syntax to assign the same value to all three variables in one code line. **a = b = c = 25**
Insert the correct keyword to make the variable x belong to the global scope.

```
def myfunc():
    global x
```
```

Data Types in Python

Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

```
myVariable = 4       # myVariable is of type int
myVariable = "Sally" # myVariable is now of type str
```

Exercise:

The following code example would print the data type of myVar, what data type would t

myVar = "Hello World" ; print(type(**myVar**))

myVar = 20.5 ; print(type(x **myVar**

myVar = ["apple", "banana", "cherry"] ; print(type(**myVar**))

myVar = (”peach", ”nectarine", ”papaya") ; print(type(**myVar**))

myVar = {"name" : ”BLUE", ”height" : 22} ; print(type(**myVar**))

myVar = False ; print(type(**myVar**))

myVar = 5

Typecast to integer

Typecast to float

Typecast to complex

Number Datatypes in Python

- There are three numeric types in Python:
- Int : Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

- Float: Float, or "floating point number" is a number, positive or negative, containing one or more decimals.
- Float can also be scientific numbers with an "e" to indicate the power of 10.
- Complex: myVar = 2.6j # complex
- Casting : There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types. int(), str(), float()

Exercise:

myVar = 5
Typecast to integer
Typecast to float
Typecast to complex

Strings in Python

Strings in python are surrounded by either single quotation marks, or double quotation marks.
'hello' is the same as "hello".

Assigning a string to a variable is done with the variable name followed by an equal sign and the string

You can assign a multiline string to a variable by using three quotes Or three single quotes

Strings are really an array of characters so you can parse them using index.
A = “Hello”
A[0] =? -> H

Slicing : You can return a range of characters by using the slice syntax. Index starts at 0
Variablename>[<start index>:<end index>]
Variablename [:<end index>] starts from the beginning
Variablename [<start index>:] ends at the end
Use negative indexes to start the slice from the end of the string:

```
myVar = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""”

mVar=  '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(myVar)
```

Strings in Python

Strings are really an array of characters so you can parse them using index.
A = “Hello”
A[0] =? -> H

Slicing : You can return a range of characters by using the slice syntax. Index starts at 0
Variablename>[<start index>:<end index>]
Variablename [:<end index>] starts from the beginning
Variablename [<start index>:] ends at the end
Use negative indexes to start the slice from the end of the string:

Modification of Strings: upper() **,** lower() **, strip(),** replace(), split()

Concatenate: To concatenate, or combine, two strings you can use the + operator. To add a space between them, add a " ”

Format Strings: format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

```
myVar = """Lorem ipsum dolor sit
amet,
consectetur adipiscing elit,
sed do eiusmod tempor
incididunt
ut labore et dolore magna
aliqua."""”

mVar=  '''Lorem ipsum dolor sit
amet,
consectetur adipiscing elit,
sed do eiusmod tempor
incididunt
ut labore et dolore magna
aliqua.'''
print(myVar)
```

Strings in Python

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Loop through the letters in the word ”lamps":

```
for myVar in ”lamps":
  print(x)
```

To get the length of a string, use the len() function.

To check if a certain phrase or character is present in a string, we can use the keyword in.

Check if ”easy" is present in the following text:

```
txt = ”Python is an easy language"
print(”easy" in txt)
```

Use it in an if statement

Print only if ”easy" is present:

```
txt = ”Python is an easy language"
print(”easy" in txt)
if ”easy" in txt:
  print("Yes, ‘easy' is present.")
```

Strings in Python - Contd

Escape Character

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a single quote inside a string that is surrounded by double quotes:

Code	Result
\'	Single Quote
\\	Backslash
\n	New Line
\r	CarriageReturn
\t	Tab
\b	Backspace

Operators in Python - Contd

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Exercise:

Print length : myVar = "Hello World" ;print()

Boolean Values Types in Python

- Booleans represent one of two values: True or False.
- bool() function allows you to evaluate any value, and give you True or False in return
- Bool("Hello")
- **Most Values are True**
- Almost any value is evaluated to True if it has some sort of content.
- Any string is True, except empty strings.
- Any number is True, except 0.
- Any list, tuple, set, and dictionary are True, except empty ones.

- Empty values, such as (), [], {}, "", the number 0, and the value None. False evaluates to False.

Exercise:

```
print(100 > 90)
```

When you run a condition in an if statement, Python returns True or False:

Print a message based on whether the condition is True or False:

```
myVar1 = 10
myVar2 = 20

if myVar2 > myVar1 :
  print(" myVar2 is greater than myVar1 ")
else:
  print(" myVar2 is not greater than myVar1")
```

Casting Types in Python

Casting : Change Data Types

•int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

•float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

•str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Python Collection (Arrays)

Python Collections (Arrays)
There are four collection data types in the Python programming language:
•**List** is a collection which is ordered and changeable. Allows duplicate members.
myList = ["apple", "banana", "cherry"]
•**Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
myTuple = ("apple", "banana", "cherry")

•**Set** is a collection which is unordered, unchangeable*, and unindexed. No duplicate members. Set *items* are unchangeable, but you can remove items and add new items.
mySet = {"apple", "banana", "cherry"}
* Set *items* are unchangeable, but you can remove and/or add items whenever you like.

•**Dictionary** is a collection which is ordered** and changeable. No duplicate members.
As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

```
myDict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(myDict)
```

**As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

•When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

Lists in Python - Contd

Lists are used to store multiple items in a single variable.
Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
Lists are created using square brackets

List items are ordered, changeable, and allow duplicate values.
List items are indexed, the first item has index [0], the second item has index [1] etc

When we say **that lists are ordered**, it means that the items have a defined order, and that order will not change.
If you add new items to a list, the new items will be placed at the end of the list.

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Since lists are indexed, lists can have items with the same value

It is also possible to use the list() constructor when creating a new list.

```
myList = [1,2,3]
print(myList)

len(myList))

myList1 = [1,11,12]
myList2 = [1, 5, 7, 9, 3]
myList3 = [True, False, False]

myList = list ((1, 5, 7, 9, 3)) # note the double round-brackets
```

Lists in Python - Contd

List items are indexed and you can access them by referring to the index number:

```
mylist = [1,2,3]
print(mylist[-1])
```

Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

```
mylist = [11,12,13,14,15,16,17]
print(thislist[3:6])
print(thislist[:3])
print(thislist[4:])
print(thislist[-4:-1])
if 1 in mylist :
  print(”1 is in the list")
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

By leaving out the start value, the range will start at the first item:

To determine if a specified item is present in a list use the in keyword:

Change Item Value

To change the value of a specific item, refer to the index number:

```
mylist[0] = 190
Print(mylist)
```

```
mylist[1:3] = [100,200]
print(mylist)
```

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values,

Lists in Python - Contd

There are a list of functions that can be used to do various manipulations to the list

mylist.append(<item>)
mylist.extend<list>)
mylist.insert(<item>)
mylist.remove(<item>) (Only first occurrence would be removed)
mylist.pop(<index>) , pop()– to remove a specific item in the list
mylist.del – Delete entire list
mylist.sort()
mylist.sort(reverse = True)

```
mylist = [1,2,3]
mylist.append(4)
mylist.insert(1, 11)
mylist2 = [10,11,12,13]
mylist.extend(mylist2)
```

Del mylist

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a *reference* to list1, and changes made in list1 will automatically also be made in list2.

```
mylist = [ 1,2,3,4,5,6]
mylist2 = mylist.copy()
mylist2 = list(mylist)
```

```
mylist1 = ["a", "b", "c"]
mylist2 = [1, 2, 3]
mylist3 = list1 + list2
```

Looping through list

- You can loop through the list items by using a for loop:
- You can also loop through the list items by referring to their index number.
- You can also loop through using while loop

- Learn more about while loops in our Python While Loops Chapter.

- **List Comprehension offers the shortest syntax for looping through lists: This comes by**
- A short hand for loop that will print all items in a list:
- List Comprehension is a powerful looping method and this is a efficient way to loop and mine information from the lists.

-
```
mylist = [1,2,3,4]
for x in mylist:
  print(x)
```

-
```
mylist = [1,2,3,4]

for i in range(len(mylist)):
  print(mylist[i])
```

-
```
mylist = [1,2,3,4]
i = 0
while i < len(mylist):
  print(mylist[i])
  i = i + 1
```

-
```
mylist = [1,2,3,4]
[print(x) for x in mylist]
```

-
```
mylist = [1,2,3,4]
newlist = []

for x in fruits:
  if  1  in x:
    newlist.append(x)

print(newlist)
```

- **mylist = [x for x in mylist]**
- **mylist = [x for x in mylist if x != 1]**
- **mylist = [x for x in range(10)]**
- **mylist = [x for x in range(10) if x < 5]**

Exercises for Loops

- Write a Python program to remove duplicates from a list.
- Write a Python program to check if a list is empty or not.
- Write a Python program to clone or copy a list.
- Write a Python program to find the list of words that are longer than n from a given list of words.
- Write a Python function that takes two lists and returns True if they have at least one common member.

Write a Python program to remove duplicates from a list

```
a = [10,20,30,20,10,50,60,40,80,50,40]
dup_items = set()
    uniq_items = []
    for x in a: if x not in dup_items:
        uniq_items.append(x)
        dup_items.add(x)
        print(dup_items)
```

Write a Python program to check if a list is empty or not.

- l = [] if not l: print("List is empty")

Write a Python program to clone or copy a list.

```
original_list = [10, 22, 44, 23, 4]
new_list = list(original_list)
print(original_list)
print(new_list)
```

Write a Python program to find the list of words that are longer than n from a given list of words.

```
def long_words(n, str):
    word_len = []
    txt = str.split(" ")
    for x in txt:
        if len(x) > n:
                word_len.append(x)
                return word_len

print(long_words(3, "The quick brown fox jumps over the lazy dog"))
```

Write a Python function that takes two lists and returns True if they have at least one common member.

```
def common_data(list1, list2):
        result = False
        for x in list1:
                for y in list2:
                        if x == y:
                                result = True
                                return result
print(common_data([1,2,3,4,5], [5,6,7,8,9]))
print(common_data([1,2,3,4,5], [6,7,8,9]))
```

If ... Else

- **Python Conditions and If statements**
- Python supports the usual logical conditions from mathematics:
- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- These conditions can be used in several ways, most commonly in "if statements" and loops.
- An "if statement" is written by using the if keyword.

```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

If ... Else

- **Short Hand If ... Else**
- If you have only one statement to execute, one for if, and one for else, you can put it all on the same line

- if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error.

```
a = 200
b = 33
C=10
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
if a > b: print("a is greater than b")
print("A") if a > b else print("B")
```

```
if a > b and c > a:
  print("Both conditions are True")
```

```
if a > b or a > c:
  print("At least one of the
conditions is True")
```

```
if b > a:
  pass
```

While Loop

- With the while loop we can execute a set of statements as long as a condition is true.

- With the break statement we can stop the loop even if the while condition is true:

- With the continue statement we can stop the current iteration, and continue with the next:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

```
i = 1
while i < 6:
  print(i)
  if i == 3:
    break
  i += 1
```

```
i = 0
while i < 6:
  i += 1
  if i == 3:
      continue
   else:
      print("i is no longer less than
6")
```

For Loops

- **Python For Loops**
- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.
- With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.
- The for loop does not require an indexing variable to set beforehand.

```
for x in "banana":
  print(x)
```

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
  elseif x == “apple”:
     print(x)
```