PYTHON

Books:

- Lutz, Mark, “Learning python”, O'Reilly Media, Inc.
- Zed A. Shaw, “Learn python the hard way”, Pearson publications
- Dierbach, Charles “Python, A Computational Problem-Solving Focus”, Wiley
- http://www.davekuhlman.org/python_book_01.html
- https://github.com/learnbyexample/Python_Basics?utm_source=devfreebooks&utm_medium=medium&utm_campaign=DevFreeBooks
- https://learnpythonthehardway.org/book/?utm_source=devfreebooks&utm_medium=medium&utm_campaign=DevFreeBooks

Other readings and relevant websites:

- https://www.python.org/
- http://www.tutorialspoint.com/python/
- https://www.codecademy.com/learn/python
- http://www.pyschools.com/
- https://www.codementor.io/learn-python-online
- https://www.w3schools.com/python/
- Python IDLE (https://www.python.org/downloads/)
- Anaconda (https://www.anaconda.com/products/individual)
- Coding Ninjas (https://www.codingninjas.com/)

Topics:

- Data Types, Variables and literals, Blocks and Syntax Rules, Operators and Expressions, Assignment Statements, Expression Statements, Multiway Branching.
- Looping, Decisions, Control Flow- Conditionals and loops, pattern designing.

- Defining Functions, Scope Rules, Global Statements, Closures, Argument Matching, Passing Arguments, Recursive Functions, Lambda Expressions.
- Lists, Indexing and Slicing, References and Copies, List Comprehension, map, filter & reduce functions.
- Searching & Sorting: Imports and Attributes, Creating Modules, Searching & Sorting, Namespaces, Reloading, Generating Random values.
- Two Dimensional Lists
- Strings and its relative methods and properties
- Tuples, Set and Dictionaries- introduction, methods and its relative properties

---

- **Data Types:**

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

- 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`

- **Variables and literal:**

A Variable is a location that is named in order to store data while the program is being run.

In a programming language, Variables are words that are used to store values of any data type.

when you create a variable, it takes up some memory space based on the value and the type you set to it. The Python interpreter allocates RAM to the variable based on its data type. The variables' values can be altered at any time during the program.

```
num = 10
numlist = [1, 3, 5, 7, 9]
str = 'Hello World'

print(num)
print(numlist)
print(str)
```

output:

```
10
[1,3,5,7,9]
Hello World
```

The data which is being assigned to the variables are called as Literal.

In Python, Literals are defined as raw data which is being assigned to the variables or constants

```
str= 'How are you, Sam?'
```

- **Blocks and Syntax Rules:**
- Python is case sensitive.
- For path specification, python uses forward slashes.
- In python, there is no command terminator so no semicolon needed.
- In one line only a single executable statement should be written.
- To write two separate executable statements in a single line, you should use a semicolon ; to separate the commands.

```
print(“hello”); print(“hi”)
```

- # used for comments.
- Blank lines in between a program are ignored by python.

• Operators and Expressions:

Arithmetic Operators:

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, and division.

Operator	Description	Syntax
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power: Returns first raised to power second	x ** y

PRECEDENCE

Parentheses, Exponentiation, Multiplication, Division, Addition, Subtraction

Comparison Operators:

Comparison of Relational operators compares the values. It either returns True or False according to the condition.(Boolean value is only returned form these, can be used in if sattements).

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y
is	x is the same as y	x is y
is not	x is not the same as y	x is not y

Logical Operators:

Logical operators perform Logical AND, Logical OR, and Logical NOT operations. It is used to combine conditional statements.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

Bitwise Operators:

Bitwise operators act on bits and perform the bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

Assignment Operators:

Assignment operators are used to assign values to the variables.

Operator	Description	Syntax
=	Assign value of right side of expression to left side operand	x = y + z
+=	Add AND: Add right-side operand with left side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand	a/=b

Operator	Description	Syntax
	and then assign to left operand	a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b a= a << b

- **Assignment Statements:**

We use Python assignment statements to assign objects to names. The target of an assignment statement is written on the left side of the equal sign (=), and the object on the right can be an arbitrary expression that computes an object.

Basic form:

Most common form.

```
student ='Geeks'
```

Tuple assignment:

```
x, y = 10, 20
```

List assignment:

This works in the same way as the tuple assignment.

```
[x, y] =[1, 3]
```

Sequence assignment:

Python assigns the items one at a time by position.

```
a, b, c ='CR7'
```

Extended Sequence unpacking:

It allows us to be more flexible in how we select portions of a sequence to assign.

```
p, *q ='Hello'
```

Multiple- target assignment:

```
x =y =75
```

Augmented assignment:

The augmented assignment is a shorthand assignment that combines an expression and an assignment.

```
x =2
x +=1  (means x=x+1)

x is 3
```

- ## Expression Statements:

GeeksforGeeks Link :https://www.geeksforgeeks.org/expressions-in-python/#:~:text=An%20expression%20is%20a%20combination,the%20precedence%20of%20its%20operators.

- ## Multiway Branching:

Multiway branching is another possibility for reducing branch penalties. With multiway branching, both the sequential and the taken paths of an unresolved conditional branch are pursued, as shown in the figure. The multiway branching requires multiple program counters (PCs) referred to as IFA1 and IFA2 in the figure.

- ## Looping,Decisions, Control Flow- Conditionals and loops:

Python has two primitive loop commands:

- while loops
- for loops

The while Loop:

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

Example

Print x as long as x is less than 6:

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

output:

1
2
3
4
5
```

The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, x, which we set to 1.

Break Statement:

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

Example

Exit the loop when x is 3:

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

Try Yourself
```

Continue Statement:

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

Example

Continue to the next iteration if x is 3:

```
x =0
while(x<6):
x+=1
  if x==3:
continue
  print(x)

Try Yourself
```

Else Statement:

With the else statement we can run a block of code once when the condition no longer is true:

Example

Print a message once the condition is false:

```
x =1
while(x<6):
print(x)
  x+=1
else:
print("x is grater than 6")

Try Yourself
```

The For Loop:

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.

```
n =4
foriinrange(0, n):
    print(i)

output:

0
1
2
3
```

For loop on all type of variables:

```
print("List Iteration")
l =["geeks", "for", "geeks"]
foriinl:
    print(i)

# Iterating over a tuple (immutable)
print("\nTuple Iteration")
t =("geeks", "for", "geeks")
foriint:
    print(i)

# Iterating over a String
print("\nString Iteration")
s ="Geeks"
foriins :
    print(i)

# Iterating over dictionary
print("\nDictionary Iteration")
d =dict()
d['xyz'] =123
d['abc'] =345
foriind :
    print("%s  %d"%(i, d[i]))

#Iterating over a set
```

```
print("\nSet Iteration")
set1 ={1,2,3,4,5,6}
foriinset1:
    print(i),
```

Iterating by the index of sequences:

We can also use the index of elements in the sequence to iterate. The key idea is to first calculate the length of the list and in iterate over the sequence within the range of this length.

```
list=["geeks", "for", "geeks"]
forindex inrange(len(list)):
    printlist[index]
```

Nested Loops:

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.
Syntax:

```
foriterator_varinsequence:
    foriterator_varinsequence:
        statements(s)
        statements(s)
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/loops-in-python/

W3school Link:

https://www.w3schools.com/python/python_for_loops.asp

CodeWithHarry Link:(video)

https://www.youtube.com/watch?v=sSyCRQx5WM0

- **Defining Functions, Arguments, Parameters:**

A function is a block of code which only runs when it is called.You can pass data, known as parameters, into a function. A function can return data as a result.

Creating a Function

In Python a function is defined using the def keyword:

```
Def func():
Print("hello from func")

func()
```

Arguments:

Information can be passed into functions as arguments. Send 1 as argument to the function():

```
Def func(x):
Print("Argument =",x)

func(1)
```

Parameters or Arguments?

The terms parameter and argument can be used for the same thing: information that are passed into a function.

Return Values:

To let a function return a value, use the return statement:

```
Def func(x):
Return 5*x
```

```
Print(func(1))

Output:

5
```

Recursion:

Python also accepts function recursion, which means a defined function can call itself.

Recursion is a common mathematical and programming concept. It means that a function calls itself. This has the benefit of meaning that you can loop through data to reach a result.

The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.

- **Scope Rules, Global Statements:**

A variable is only available from inside the region it is created. This is called **scope**.

Local Scope:

A variable created inside a function belongs to the *local scope* of that function, and can only be used inside that function.

Global Scope:

A variable created in the main body of the Python code is a global variableand belongs to the global scope.Global variables are available from within any scope, global and local.

Global Keyword:

If you need to create a global variable, but are stuck in the local scope, you can use the global keyword.The global keyword makes the variable global.

- ## Closures in Python?

Like nested loops, we can also nest functions. That said, Python gives us the power to define functions within functions.

Python Closures are these inner functions that are enclosed within the outer function. Closures can access variables present in the outer function scope. It can access these variables even after the outer function has completed its execution.

- ## Python Lambda Functions:

Python Lambda Functions are anonymous function means that the function is without a name. As we already know that the def keyword is used to define a normal function in Python.

Python Lambda Function Syntax:

lambda arguments: expression

GeekforGeeks Link:

https://www.geeksforgeeks.org/python-lambda-anonymous-functions-filter-map-reduce/

W3school Link:

https://www.w3schools.com/python/python_lambda.asp

- ## Lists, Indexing and Slicing, References and Copies, List Comprehension, map, filter & reduce functions.

Lists are just like dynamically sized arrays, declared in other languages (vector in C++ and ArrayList in Java). In a simple language, a list is a

collection of things, enclosed in [ ] and separated by commas. Lists are the simplest containers that are an integral part of the Python language. Lists need not be homogeneous always which makes it the most powerful tool in Python. A single list may contain DataTypes like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

Creating a list:

```
# Creating a List
List=[]
print("Blank List: ")
print(List)
```

Length of list:

```
# Creating a List
List1 =[1,2,4]
print(len(List1))
```

Adding element to list:

Using append() method

Element be added to the List by using the built-in append() function. Only one element at a time can be added to the list by using the append() method, for the addition of multiple elements with the append() method, loops are used.

Using insert() method

append() method only works for the addition of elements at the end of the List, for the addition of elements at the desired position, insert() method is used. Unlike append() which takes only one argument, the insert() method requires two arguments(position, value).

Using extend() method

Other than append() and insert() methods, there's one more method for the Addition of elements, extend(), this method is used to add multiple elements at the same time at the end of the list.

GeekforGeeks link:

https://www.geeksforgeeks.org/python-list/#ksl

Javatpoint link:

https://www.javatpoint.com/python-lists

- **Imports in python:**

Import in python is similar to #include header_file in C/C++. Python modules can get access to code from another module by importing the file/function using import. The import statement is the most common way of invoking the import machinery, but it is not the only way.

import module_name
When the import is used, it searches for the module initially in the local scope by calling __import__() function. The value returned by the function is then reflected in the output of the initial code.

```
importmath
print(math.pi)
```

- **Python Modules:**

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

Create a Module:

To create a module just save the code you want in a file with the file extension .py:

```
Def func(name):
Print("hello, "+name)
```

Use a Module:

Now we can use the module we just created, by using the import statement:

Example

Import the module named mymodule, and call the greeting function:

```
Import mymodule

Mymodule.func("Akshit")
```

W3School Link:

https://www.w3schools.com/python/python_modules.asp

Geeksforgeeks Link:

https://www.geeksforgeeks.org/create-and-import-modules-in-python/

- ## Namespaces:

A namespace is a system that has a unique name for each and every object in Python. An object might be a variable or a method. Python itself maintains a namespace in the form of a Python dictionary. Let's

go through an example, a directory-file system structure in computers. Needless to say, that one can have multiple directories having a file with the same name inside every directory. But one can get directed to the file, one wishes, just by specifying the absolute path to the file.

Types of namespaces:

When Python interpreter runs solely without any user-defined modules, methods, classes, etc. Some functions like print(), id() are always present, these are built-in namespaces. When a user creates a module, a global namespace gets created, later the creation of local functions creates the local namespace. The built-in namespace encompasses the global namespace and the global namespace encompasses the local namespace.

The lifetime of a namespace:

A lifetime of a namespace depends upon the scope of objects, if the scope of an object ends, the lifetime of that namespace comes to an end. Hence, it is not possible to access the inner namespace's objects from an outer namespace.

Scope of Objects in Python:

Scope refers to the coding region from which a particular Python object is accessible. Hence one cannot access any particular object from anywhere from the code, the accessing has to be allowed by the scope of the object.
Let's take an example to have a detailed understanding of the same:

- **Reloading modules in Python?**

The reload() - reloads a previously imported module or loaded module. This comes handy in a situation where you repeatedly run a test script during an interactive session, it always uses the first version of the

modules we are developing, even we have made changes to the code. In that scenario we need to make sure that modules are reloaded.

The argument passed to the reload() must be a module object which is successfully imported before.

Syntax

Import importlib

Importlib.reload(sys)

- **Generating random number list in Python:**

There is a need to generate random numbers when studying a model or behaviour of a program for different range of values. Python can generate such random numbers by using the random module. In the below examples we will first see how to generate a single random number and then extend it to generate a list of random numbers.

The random() method in random module generates a float number between 0 and 1.

Example

```
import random
n = random.random()
print(n)
```

Using random.sample():

We can also use the sample() method available in random module to directly generate a list of random numbers.Here we specify a range and give how many random numbers we need to generate.

```
import random
list=random.sample(range(10, 30), 5)
print(list)
```

- **2D arrays Python:**

Python provides many ways to create 2-dimensional lists/arrays. However one must know the differences between these ways because they can create complications in code that can be very difficult to trace out. Lets start by looking at common ways of creating 1d array of size N initialized with 0s.

```
rows, cols =(5, 5)
arr=[[0]*cols]*rows
print(arr)

output:

[[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]]
```

Geeksforgeeks Link:

https://www.geeksforgeeks.org/python-using-2d-arrays-lists-the-right-way/

snakify.org Link:

https://snakify.org/en/lessons/two_dimensional_lists_arrays/

- **String in python:**

Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Creating a String:

Strings in Python can be created using single quotes or double quotes or even triple quotes.

```
# Creating a String
# with single Quotes
String1 ='Welcome to the Geeks World'
print("String with the use of Single Quotes: ")
print(String1)

# Creating a String
# with double Quotes
String1 ="I'm a God"
print("\nString with the use of Double Quotes: ")
print(String1)
```

Accessing characters in Python:

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character, and so on.

```
String1 ="helloMyFriend"
print("Initial String: ")
print(String1)
```

String Slicing:

To access a range of characters in the String, the method of slicing is used. Slicing in a String is done by using a Slicing operator (colon).

```
# Creating a String
String1 ="saywhat"
print("Initial String: ")
print(String1)
```

Deleting/Updating from a String:

In Python, Updation or deletion of characters from a String is not allowed. This will cause an error because item assignment or item

deletion from a String is not supported. Although deletion of the entire String is possible with the use of a built-in del keyword. This is because Strings are immutable, hence elements of a String cannot be changed once it has been assigned. Only new strings can be reassigned to the same name.

```
String1 ="Hello, I'm a Geek"
print("Initial String: ")
print(String1)
```

GeeksforGeeks Link:

https://www.geeksforgeeks.org/python-strings/

W3school Link:

https://www.w3schools.com/python/python_strings.asp#:~:text=Strings%20are%20Arrays,access%20elements%20of%20the%20string.

- **Tuple:**

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

Example

Create a Tuple:

```
Ele=(“apple”, “banana”, “cherry”)
Print(ele)
```

Tuple Items:

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered:

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable:

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

W3school Links:

https://www.w3schools.com/python/python_tuples.asp#:~:text=%2C%20%22cherry%22)-,Tuple,which%20is%20ordered%20and%20unchangeable.

Lists	Tuples	Sets	Dictionaries
A list is a collection of ordered data.	A tuple is an ordered collection of data.	A set is an unordered collection.	A dictionary is an unordered collection of data that stores data in key-value pairs.

Various ways to create a list:

```
list1=[1,4,"Gitam",6,"college"]
list2=[]  # creates an empty list
list3=list((1,2,3))
print(list1)
print(list2)
print(list3)
```

Various ways to create a tuple:

```
tuple1=(1,2,"college",9)
tuple2=()# creates an empty tuple
tuple3=tuple((1,3,5,9,"hello"))
print(tuple1)
print(tuple2)
print(tuple3)
```

How to create a set:

```
set1={1,2,3,4,5,"hello","tup"}
set2={(1,8,"python",7)}
print(set1)
print(set2)
```

How to create a dictionary:

```
dict1={"key1":"value1","key2":"value2"}
dict2={}   # empty dictionary
dict3=dict({1:"apple",2:"cherry",3:"strawberry"})
print(dict1)
print(dict2)
print(dict3)
```

Lists are mutable.	Tuples are immutable.	Sets are mutable and have no duplicate elements.	Dictionaries are mutable and keys do not allow duplicates.
Lists are declared with square braces.	Tuples are enclosed within parenthesis.	Sets are represented in curly brackets.	Dictionaries are enclosed in curly brackets in the form of key-value pairs.

- **Link for practice website:**

Dsacamp link:

https://dsacamp.in/dsa
