

OBJECTIVE:

INTRODUCTION TO PYTHON

COLAB:

Colaboratory by Google (Google Colab in short) is a Jupyter notebook based runtime environment which allows you to run code entirely on the cloud.

This is necessary because it means that you can train large scale ML and DL models even if you don't have access to a powerful machine or a high speed internet access.

Google Colab supports both GPU and TPU instances, which makes it a perfect tool for deep learning and data analytics enthusiasts because of computational limitations on local machines.

Since a Colab notebook can be accessed remotely from any machine through a browser, it's well suited for commercial purposes as well.

Creating your first .ipynb notebook in colab

Open a browser of your choice and go to colab.research.google.com and sign in using your Google account. Click on a new notebook to create a new runtime instance.

```
[1] print("Hello World")

Hello World

[2] inp1 = int(input())
inp2 = int(input())

add = inp1 + inp2
print(add)

5
5
10

inp1 = int(input())
inp2 = int(input())

5
5
```

Pros:

- Pre-built with lots of python library
- Quick Start of python learning
- No infra setup required
- No charges for GPU usage
- Can run your code for 24 hrs without interruptions but not more than that
- Your notebooks is saved in google drive only

Cons:

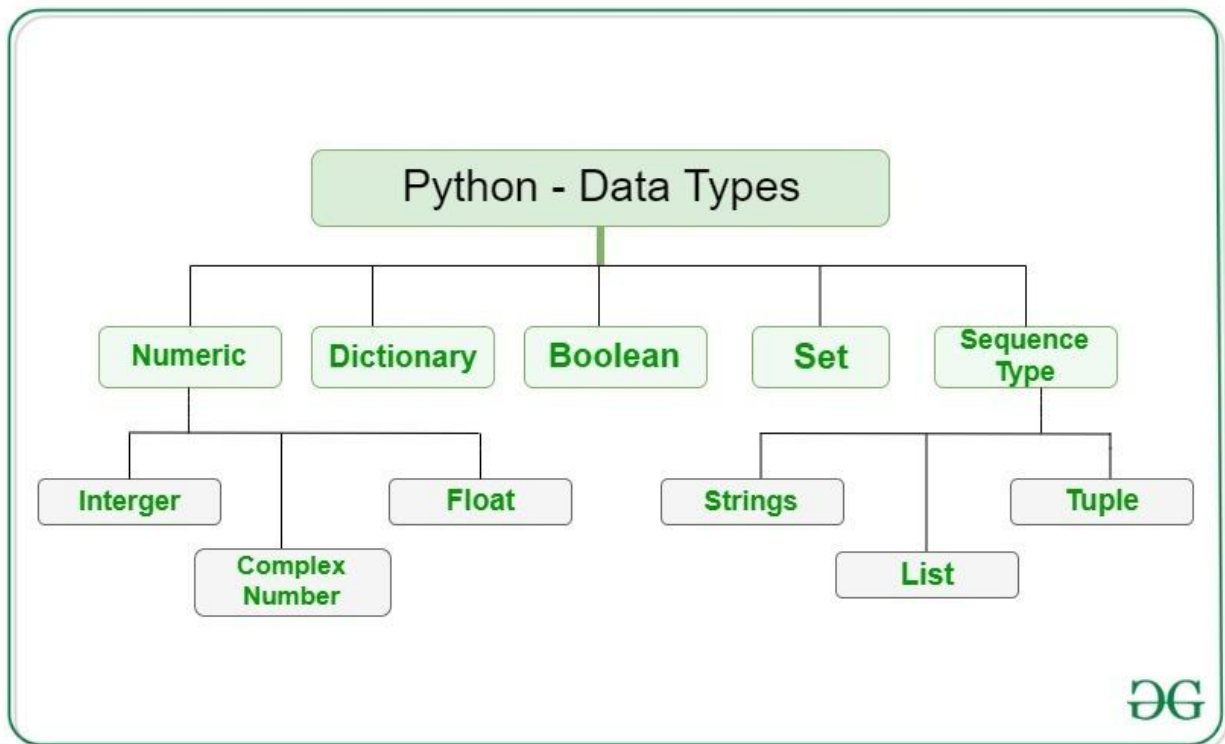
- Need to install all specific libraries which does not come with standard python (Need to repeat this with every session)
- Google Drive is your source and target for Storage, there are other like local (which eats your bandwidth if dataset is big)
- Google provided the code to connect and use with google drive but that will not work with lots of other data format
- Google Storage is used with current session, so if you have downloaded some file and want to use it later, better save it before closing the session.
- Difficult to work with BIGGER datasets as you have to download and store them in Google drive (15 GB Free space with gmail id, additional required a payment towards google)

DATA TYPES IN PYTHON:

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, data types are actually classes and variables are instances (object) of these classes.

Following are the standard or built-in data type of Python:

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary



1) Numeric

In Python, numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part)j*. For example – $2+3j$

Note – type() function is used to determine the type of data type.

2) Sequence Type

In Python, sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

A) String

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Creating String

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

B) List

Lists are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Creating List

Lists in Python can be created by just placing the sequence inside the square brackets[].

C) Tuple

Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by tuple class.

Creating Tuple

In Python, tuples are created by placing a sequence of values separated by 'comma' with or without the use of parentheses for grouping of the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, list, etc.).

Note: Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing 'comma' to make it a tuple.

Note – Creation of Python tuple without the use of parentheses is known as Tuple Packing.

3) Boolean

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). But non-Boolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.

Note – True and False with capital ‘T’ and ‘F’ are valid booleans otherwise python will throw an error.

4) Set

In Python, Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Creating Sets

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by ‘comma’. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

5) Dictionary

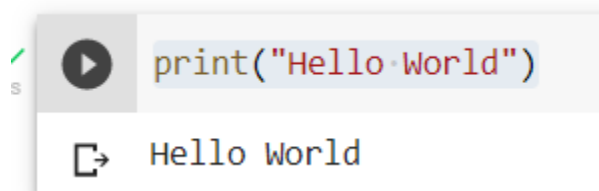
Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a ‘comma’.

Creating Dictionary

In Python, a Dictionary can be created by placing a sequence of elements within curly {} braces, separated by ‘comma’. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable. Dictionary can also be created by the built-in function dict(). An empty dictionary can be created by just placing it to curly braces{ }.

Note – Dictionary keys are case sensitive, same name but different cases of Key will be treated distinctly.

A) HELLO WORLD:

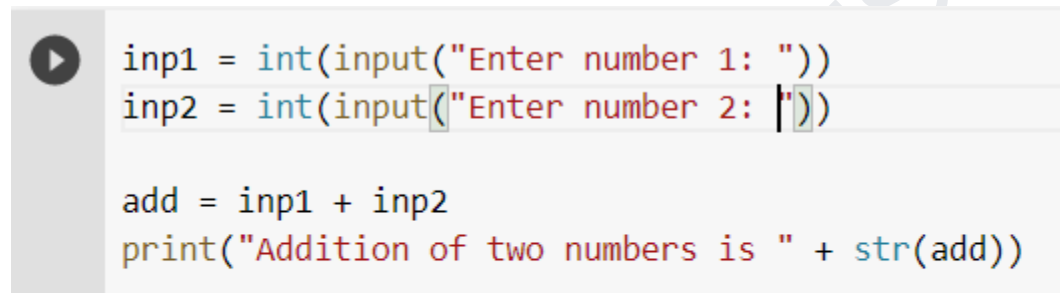


```
print("Hello World")
```

Output: Hello World

B) ARITHMETIC OPERATIONS:

1) ADDITION:

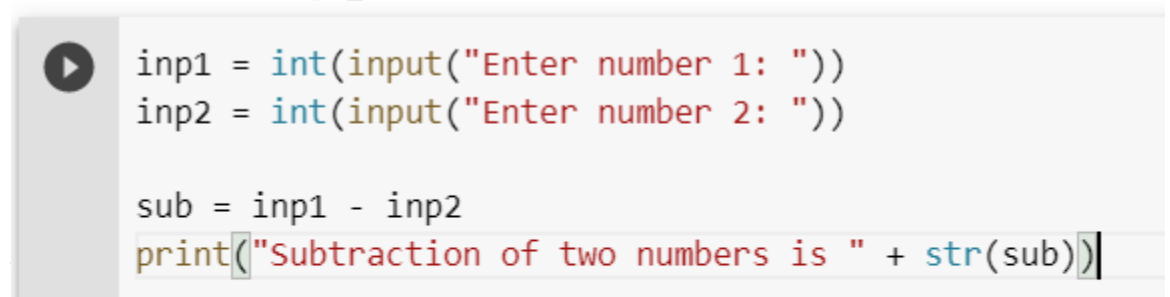


```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))

add = inp1 + inp2
print("Addition of two numbers is " + str(add))
```

Enter number 1: 5
Enter number 2: 10
Addition of two numbers is 15

2) SUBTRACTION:



```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))

sub = inp1 - inp2
print("Subtraction of two numbers is " + str(sub))
```

Enter number 1: 10
Enter number 2: 5
Subtraction of two numbers is 5

3) MULTIPLICATION:

```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))
mul = inp1 * inp2
print("Multiplication of two numbers is " + str(mul))
```

```
Enter number 1: 5
Enter number 2: 5
Multiplication of two numbers is 25
```

4) DIVISION:

```
inp1 = int(input("Enter number 1: "))
inp2 = int(input("Enter number 2: "))
div = inp1 / inp2
print("Division of two numbers is " + str(div))
```

```
Enter number 1: 10
Enter number 2: 5
Division of two numbers is 2.0
```

C) AREA OF CIRCLE:

```
import math
radius = int(input("Enter radius for circle: "))
areaOfCircle = math.pi * radius * radius
print("Area of Circle is ",areaOfCircle)
```

```
Enter radius for circle: 5
Area of Circle is 78.53981633974483
```


D) AREA OF TRIANGLE:

```
▶ b = int(input("Enter breadth for triangle: "))  
h = int(input("Enter height for triangle: "))  
|  
areaOfTriangle = 0.5 * b * h  
print("Area of Triangle is ",areaOfTriangle)
```

```
↳ Enter breadth for triangle: 5  
Enter height for triangle: 10  
Area of Triangle is 25.0
```

E) EVEN NUMBERS:

```
▶ evenTerms = int(input(""))  
print("Following are the even terms from 0 to", str(evenTerms))  
for i in range((evenTerms*2)+1):  
    if(i%2)==0:  
        print(i)
```

```
10  
Following are the even terms from 0 to 10  
0  
2  
4  
6  
8  
10  
12  
14  
16  
18  
20
```

F) ODD NUMBERS:

```
▶ oddTerms = int(input())
print("Following are the odd terms from 0 to", str(oddTerms))
for i in range((oddTerms*2)+1):
    if(i%2)!=0:
        print(i)
```

10
Following are the odd terms from 0 to 10
1
3
5
7
9
11
13
15
17
19

G) GENERATE MULTIPLICATION TABLE USING LOOPS:

```
▶ print("Multiplication Table:")
tableToPrint = int(input("Enter number "))
for i in range(11):
    print(str(tableToPrint) + " x " + str(i) + " = " + str(i * tableToPrint))
```

☞ Multiplication Table:
10
10 x 0 = 0
10 x 1 = 10
10 x 2 = 20
10 x 3 = 30
10 x 4 = 40
10 x 5 = 50
10 x 6 = 60
10 x 7 = 70
10 x 8 = 80
10 x 9 = 90
10 x 10 = 100

H) WHILE LOOP:

WHILE LOOP

```
▶ num_of_times = 1

while num_of_times <= 7:
    print("Narender Keswani")
    num_of_times += 1
```

```
↳ Narender Keswani
   Narender Keswani
   Narender Keswani
   Narender Keswani
   Narender Keswani
   Narender Keswani
   Narender Keswani
```

I) PYRAMID:

```
[ ] rows = int(input("Enter number of rows: "))

for i in range(rows):
    for j in range(i+1):
        print("Narender ", end="")
    print("\n")
```

```
Enter number of rows: 5
Narender

Narender Narender

Narender Narender Narender

Narender Narender Narender Narender

Narender Narender Narender Narender Narender
```

J) PRIME NUMBER CHECK:

10 PRIME NUMBER CHECK

```
[ ] num = int(input("Enter a number: "))

flag = False

if num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            flag = True
            break

if flag:
    print(num, "is not a prime number")
else:
    print(num, "is a prime number")
```

```
Enter a number: 5
5 is a prime number
```

K) GENERATE PRIME NUMBERS:

11 GENERATE PRIME NUMBERS

```
[1] lower_value = int(input ("Please, Enter the Lowest Range Value: "))
    upper_value = int(input ("Please, Enter the Upper Range Value: "))

    print ("The Prime Numbers in the range are: ")
    for number in range (lower_value, upper_value + 1):
        if number > 1:
            for i in range (2, number):
                if (number % i) == 0:
                    break
            else:
                print (number)
```

```
Please, Enter the Lowest Range Value: 5
Please, Enter the Upper Range Value: 15
The Prime Numbers in the range are:
5
7
11
13
```

L) PALINDROME NUMBER CHECK:

12 PALINDROME NUMBER CHECK

```
▶ def isPalindrome(s):  
    return s == s[::-1]  
  
s = "malayalam"  
ans = isPalindrome(s)  
|  
if ans:  
    print("Yes")  
else:  
    print("No")
```

Yes

M) TAKE INPUT FROM USER(ROLL, NAME, DOB, ADDRESS) AND STORE IN ARRAY DATATYPE

13 TAKE INPUT FROM USER (ROLL, NAME, DOB, ADDRESS) AND STORE IN ARRAY DATATYPE

```
roll = int(input("Enter roll "))
name = input("Enter name ")
dob = input("Enter dob ")
address = input("Enter address ")

thisdict = {
    "roll": roll,
    "name": name,
    "dob": dob,
    "address": address
}

for x, y in thisdict.items():
    print("The value of ", x, "is ", y)
```

Enter roll 24
Enter name Narender Keswani
Enter dob 10/11/1999
Enter address Ulhasnagar
The value of roll is 24
The value of name is Narender Keswani
The value of dob is 10/11/1999
The value of address is Ulhasnagar

PYTHON EXERCISE CRASH COURSE:

A) What is 7 to the power of 4?

```
7**4  
  
2401
```

B) Split this string

```
[2] s = "Hi there Narender Keswani!"  
    x = s.split()  
    print(x)  
  
['Hi', 'there', 'Narender', 'Keswani!']
```

C) Use .format() to print the following string

```
✓ [13] planet = "Earth"  
0s     diameter = 12742
```

```
✓ [14] print("The diameter of {} is {} kilometers.".format(planet, diameter))  
0s
```

The diameter of Earth is 12742 kilometers.

D) Given this nested list, use indexing to grab the word "hello"

```
✓ [7] lst = [1,2,[3,4],[5,[100,200,['hello']],23,11],1,7]  
0s
```

```
✓ [11] print(lst[3][1][2])  
0s
```

['hello']

E) Given this nested dictionary grab the word "hello". Be prepared, this will be annoying/tricky

```
[19] d = {'k1':[1,2,3,{'tricky':['oh','man','inception',{'target':[1,2,3,'hello']}]]}]
      [19] d['k1'][3]['tricky'][3]['target'][3]
      'hello'
```

F) Difference between list and tuple:

| SR.NO. | LIST | TUPLE |
|--------|---|---|
| 1 | Lists are mutable | Tuples are immutable |
| 2 | Implication of iterations is Time-consuming | The implication of iterations is comparatively Faster |
| 3 | The list is better for performing operations, such as insertion and deletion. | Tuple data type is appropriate for accessing the elements |
| 4 | Lists consume more memory | Tuple consume less memory as compared to the list |
| 5 | Lists have several built-in methods | Tuple does not have many built-in methods. |
| 6 | The unexpected changes and errors are more likely to occur | In tuple, it is hard to take place. |

G) Create a function that grabs the email website domain from a string in the form

```
[22] def domainGet(email):  
      a = email.split('@')  
      return a[1]  
  
[23] domainGet('user@domain.com')  
  
      'domain.com'
```

H) Create a basic function that returns True if the word 'dog' is contained in the input string. Don't worry about edge cases like a punctuation being attached to the word dog, but do account for capitalization.

```
[39] def findDog(dog):  
      if "Dog" in dog or "dog" in dog:  
          return True  
      else:  
          return False  
  
      findDog('Is there a dog here?')  
  
      True
```

I) Create a function that counts the number of times the word "dog" occurs in a string. Again ignore edge cases.

```
[31] def countDog(c):  
      return c.count('dog')  
  
[32] countDog('This dog runs faster than the other dog dude!')
```

J) Use lambda expressions and the filter() function to filter out words from a list that don't start with the letter 's'.

```
[33] seq = ['soup', 'dog', 'salad', 'cat', 'great']
```

```
[35] list(filter(lambda item: item.startswith('s'), seq))
```

```
['soup', 'salad']
```

K) You are driving a little too fast, and a police officer stops you. Write a function to return one of 3 possible results: "No ticket", "Small ticket", or "Big Ticket". If your speed is 60 or less, the result is "No Ticket". If speed is between 61 and 80 inclusive, the result is "Small Ticket". If speed is 81 or more, the result is "Big Ticket". Unless it is your birthday (encoded as a boolean value in the parameters of the function) -- on your birthday, your speed can be 5 higher in all cases.

```
✓ [36] def caught_speeding(speed, is_birthday):  
0s     if is_birthday:  
        speed = speed - 5  
     if speed > 80:  
         return 'Big Ticket'  
     elif speed > 60:  
         return 'Small Ticket'  
     else:  
         return 'No Ticket'
```

```
✓ [37] caught_speeding(81, True)  
0s  
      'Small Ticket'
```

```
✓ [38] caught_speeding(81, False)  
0s  
      'Big Ticket'
```