Python Introduction

Installations

Installation: 1) Download and install python from python.org 2) Download and install PyCharm from jetbrains.com/pycharm 3) Install jupyter-lab from command prompt using the command >>pip install jupyterlab

Initialisation: 1) Open jupyter lab from shell command prompt using the command >>jupyter-lab

Note: 1) All jupyter notebook files are stored with .jynb extension 2) Unlike python code editor, the notebook editor shows output (of the last code line only) even without print

Getting Started

In [147…
```
12343445.12 # int and float
2+4
```

Out[147]:
```
6
```

In [148…
```
"I am jupyter notebook 345978 @#$"
```

Out[148]:
```
'I am jupyter notebook 345978 @#$'
```

In [4]:
```
6+3
10*5-2
```

Out[4]:
```
48
```

In [8]:
```
print(6+3)
print(10*5-2)
print(32/5+2.1)
```

```
9
48
8.5
```

In [152…
```
type(132)
```

Out[152]:
```
int
```

> **Basic data types** in python: int, float, string (str), boolean, etc.
>
> Strings can be defined using Single-quotes or Double-quotes

In [154…
```
print("Hello World!")
print("I'm a simple print statement.")
print(3 > 5)
```

```
Hello World!
I'm a simple print statement.
False
```

Variables types and operators:

In [11]:
```
name = "Nitin Thokare"
age = 20
height = 10.5
favorite_sport = "Racing"
```

In [28]:
```
print(name)
print(type(name))
print(age)
print(type(age))
```

```
Nitin Thokare
<class 'str'>
20
<class 'int'>
```

Basic arithmetic operations

In [49]:
```
a = 20
b = 25
print(a+b)
print(b/a)
print(a**3) # power
print(25//4) # integer part
print(25%4) # remainder
```

```
45
1.25
8000
6
1
```

Comparison operators

In [156…
```
a = 20
b = 25
print(a<b)
print(a>b)
print(a==b)
print(a!=b)
print(20 >= 15)
# print(10=20) # error
# = is an assignment operator
```

```
True
False
False
True
True
```

Logical operators:

In [158…
```
print(10<12 or 'a'=='b')
print(10<12 and 'a'=='b')
print(not 'a'=='b')
```

```
True
False
True
```

Combining String with other variables:

In [162…
```
name = "Nitin Thokare"
age = 20
print("My name is "+name) # "My name isNitin Thokare"
print("My name is",name)
print("I'm",age,"years old.")

# Note the extra space in first print
```

```
My name is Nitin Thokare
My name is Nitin Thokare
I'm 20 years old.
```

f-strings

In [165…
```
line = f"My name is {name}."
print(line)
age = 22
print(f"I'm {age} years old.")
```

```
My name is Nitin Thokare.
I'm 22 years old.
```

In [18]:
```
is_student = True
is_employed = False
print(is_student)
print(f"Am I employed?: {is_employed}")
```

```
True
Am I employed?: False
```

Taking input from user:

In [167…
```
saving = input("What is your current saving?: ")
credit = input("How much you have credited?: ")
print(type(saving))
print(f"Your old saving was {saving}.\nYour new saving is {saving+credit}.")
```

```
<class 'str'>
Your old saving was 2300.
Your new saving is 2300100.
```

Type conversion

> Above code took the input as strings and hence + operation printed the numbers concatenated as strings
>
> To use them as numbers, we need to convert their types.

In [168…
```
saving = input("What is your current saving?: ")
print(type(saving))
saving = int(saving)
print(type(saving))
credit = int(input("How much you have credited?: "))
print(f"Your old saving was {saving}.\nYour new saving is {saving+credit}.")
```

```
<class 'str'>
<class 'int'>
```

```
Your old saving was 2300.
Your new saving is 2500.
```

In [203…
```python
n = float("13.5")
print(n*2)
n1 = "abcd"
print(5*n1)
```

```
27.0
abcdabcdabcdabcdabcd
```

indexing, negative indexing, length of string

In [172…
```python
s = "This is a sample string."
print(s)
print(s[0]) # T
print(s[2]) # i
print(s[8]) # a
print(s[3:13]) # s is a sam
print(f"length of the string is {len(s)}.")
print(s[-1]) # .
print(s[-4]) # i
print(s[-(len(s)-1)]) # h
```

```
This is a sample string.
T
i
a
s is a sam
length of the string is 24.
.
i
h
```

Operations on strings:

In [175…
```python
s = "This is a sample string."
print(s)
print(s.upper())
# s = s.upper()
print(s.lower())
print(s.title())
print(s)
```

```
This is a sample string.
THIS IS A SAMPLE STRING.
this is a sample string.
This Is A Sample String.
This is a sample string.
```

In [176…
```python
# find
print(s.find('a'))
print(s.find('b'))
print(s.find('Sample'))
print(s.find('sample'))
```

```
8
-1
-1
10
```

In [177…
```python
# replace
s = "This is a sample string."
print(s.replace('is', 'are'))
```

```
print(s) # original string remain unchanged
print(s.replace('Is', 'are')) # not found due to case mismatch
```

```
Thare are a sample string.
This is a sample string.
This is a sample string.
```

In [178…
```
# in
# in keyword check for availability of one string into another
s = "This is a sample string."
print('is' in s)
print('in' in s)
print('are' in s)
```

```
True
True
False
```

Conditional statements

if-elif-else

In [180…
```
name = 'nitin'
# name = 'thokare'
if name == 'nitin':
    print("Valid person. You can enter into the house.")
    print("Another line.")
    a = 2+4
else:
    print("Invalid person! You are not allowed to enter the house.")
```

```
Invalid person! You are not allowed to enter the house.
```

In [182…
```
age = 3 # in years
if age <= 1:
    print("You are a Baby!")
elif age <= 3:
    print("You are a Toddler!")
elif age <= 5:
    print("You are a Preschool child!")
elif age <= 12:
    print("You are a Gradeschool child!")
elif age <= 18:
    print("You are a Teen!")
elif age <= 21:
    print("You are a Young Adult!")
elif age <= 60:
    print("You are an Adult!")
else:
    print("You are in your Old Age!")
```

```
You are a Toddler!
```

In [186…
```
year = 1900
is_leap_year = False
if year%100 == 0:
    if year%400 == 0:
        is_leap_year = True
    else:
        is_leap_year = False
elif year%4 == 0:
    is_leap_year = True
else:
```

```
    is_leap_year = False

if is_leap_year:
    print(f"{year} is a leap year.")
else:
    print(f"{year} is not a leap year.")
```

```
1900 is not a leap year.
```

Assignment:

> **Build a simple calculator** that takes two numbers as input (say fn and sn) and an operator (+, -, *, /, **, etc.).
>
> Perform the operation between the two numbers as per the operator input and print the result as the output
>
> Show error message if the operator or any input is not valid

Range in python

> Syntax: range(start=0, end, step=1)

In [188…
```
n = range(7)
# it returns a sequence of numbers from 0 to 7-1 (i.e. 7 not included.
print(n)
print(list(n)) # more on list later

n = range(5,18,5)
print(list(n))
```

```
range(0, 7)
[0, 1, 2, 3, 4, 5, 6]
[5, 10, 15]
```

Looping in python

In [190…
```
#e.g. Print the table of 5:
print(5*1, end="; ")
print(5*2, end="; ")
print(5*3, end="; ")
print(5*4, end="; ")
print(5*5, end="; ")
print(5*6, end="; ")
print(5*7, end="; ")
print(5*8, end="; ")
print(5*9, end="; ")
print(5*10)
```

```
5; 10; 15; 20; 25; 30; 35; 40; 45; 50
```

print table using loops

While loops:

In [191…
```
i = 1
while i<=10:
```

```
    print(5*i, end="; ")
    i = i+1
print("End of while.")
```

```
5; 10; 15; 20; 25; 30; 35; 40; 45; 50; End of while.
```

For loops:

In [192…
```
for i in range(1,11):
    print(5*i, end="; ")
print("End of for loop.")
```

```
5; 10; 15; 20; 25; 30; 35; 40; 45; 50; End of for loop.
```

In [227…
```
# shorthand notation of for loop
[i for i in range(1,11)]
```

Out[227]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

In [85]:
```
# Multiplication with string
i = 1
while i<=10:
    print(i*'@ ')
    i = i+1
```

```
@
@ @
@ @ @
@ @ @ @
@ @ @ @ @
@ @ @ @ @ @
@ @ @ @ @ @ @
@ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @
@ @ @ @ @ @ @ @ @ @
```

Assignment:

> Using while and for loop, print a triangular flag of stars as below.
>
> ```
> **********
> ********
> ******
> ****
> **
> *
> *
> *
> ```

Lists:

- It is collection of data. Data can be of same or different types.
- It is represented by square brackets [ ] with elements separated by comma

In [209…
```
prices = [120, 48, 230, 34, 20, 182]
print(prices[0])
print(prices[2])
print(prices[-1])
print(prices[-3])
```

```
120
230
182
34
```

Methods on list:

In [210…
```
print("Before insert:",prices)
print("No. of elements in the list:",len(prices))
prices.insert(1, 300) # [120, 300, 48, 230, 34, 20, 182]
print("After insert:",prices)
print("No. of elements in the list:",len(prices))
prices.append(400)
print(prices)
```

```
Before insert: [120, 48, 230, 34, 20, 182]
No. of elements in the list: 6
After insert: [120, 300, 48, 230, 34, 20, 182]
No. of elements in the list: 7
[120, 300, 48, 230, 34, 20, 182, 400]
```

In [211…
```
prices.clear()
print(prices)
```

```
[]
```

Other methods on list:

- list.append(x) # append x to end of list
- list.extend(iterable) # append all elements of iterable to list
- list.insert(i, x) # insert x at index i
- list.remove(x) # remove first occurance of x from list
- list.pop([i]) # pop element at index i (defaults to end of list)
- list.clear() # delete all elements from the list
- list.index(x[, start[, end]]) # return index of element x
- list.count(x) # return number of occurances of x in list
- list.reverse() # reverse elements of list in-place (no return)
- list.sort(key=None, reverse=False) # sort list in-place
- list.copy() # return a shallow copy of the list

In [214…
```
num1 = [1, 3, 5, 7, 9]
num2 = num1.copy()
print(num2)
num2.append(11)
print(num2)
print(num1)
```

```
[1, 3, 5, 7, 9]
[1, 3, 5, 7, 9, 11]
[1, 3, 5, 7, 9]
```

Looping on list

In [215…
```
num = [1, 3, 5, 7, 9]
for i in num:
    print(10*i, end="; ")
print("End.")
```

```
10; 30; 50; 70; 90; End.
```

break & continue in loop

- **break** is used to come out of the loop before it ends by itself
- **continue** is used to skip execution of further statements in the loop

In [216…
```
num = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30]
for a in num:
    if 10 < a < 16:
        continue
    if a > 24:
        break
    print(a, end="; ")
print("Exit done.")
```

```
2; 4; 6; 8; 10; 16; 18; 20; 22; 24; Exit done.
```

Tuple:

- It is immutable (cannot be modified)
- It is represented by circular brackets ( )

In [107…
```
marks = (73, 82, 91, 78, 73, 91, 91, 80)
print(marks)
print(type(marks))
print(marks[0])
print(marks[3])
print(marks[-1])
print(marks[-2])
```

```
(73, 82, 91, 78, 73, 91, 91, 80)
<class 'tuple'>
73
78
80
91
```

In [114…
```
# list without brackets is considered as a tuple
num = 20, 21, 22, 23
print(num)
print(type(num))
```

```
(20, 21, 22, 23)
<class 'tuple'>
```

In [113…
```
# marks[0] = 10 # error
```

Methods on tuple

- count(x) : Returns the number of times 'x' occurs in a tuple
- index(x) : Searches the tuple for 'x' and returns the position of where it was first found

In [115…
```
print(marks.count(91))
print(marks.index(78))
```

```
3
3
```

Set:

- It is a collection of unique elements (no duplicates are stored)
- It is represented by { }

In [116…
```
ids = {12, 13, 14, 15, 15}
print(ids)
print(type(ids))
```

```
{12, 13, 14, 15}
<class 'set'>
```

In [119…
```
ids.add(12)
print(ids)
ids.add(16)
print(ids)
```

```
{12, 13, 14, 15}
{12, 13, 14, 15, 16}
```

In [123…
```
# indexing is not applicable on set.
# Elements are stored in any order (they are unordered)

# print(ids[0]) # error: 'set' object is not subscriptable
```

Methods on set:

- add() #Adds an element to the set
- clear() #Removes all the elements from the set
- copy() #Returns a copy of the set
- difference() #Returns a set containing the difference between two or more sets
- difference_update() #Removes the items in this set that are also included in another, specified set
- discard() #Remove the specified item
- intersection() #Returns a set, that is the intersection of two or more sets
- intersection_update() #Removes the items in this set that are not present in other, specified set(s)
- isdisjoint() #Returns whether two sets have a intersection or not
- issubset() #Returns whether another set contains this set or not
- issuperset() #Returns whether this set contains another set or not
- pop() #Removes an element from the set
- remove() #Removes the specified element
- symmetric_difference() #Returns a set with the symmetric differences of two sets
- symmetric_difference_update() #inserts the symmetric differences from this set and another
- union() #Return a set containing the union of sets
- update() #Update the set with another set, or any other iterable

Dictionary:

- It is a collection of key-value pairs.
- Keys are unique in the dictionary, value can be repeated
- They are represented like { key1 : value1, key2 : value2 }

In [217…
```
marks = {"physics": 81, "chemistry": 85, "maths": 78, "english":81} # roll_number: name
print(marks["physics"]) # access by key
# print(marks[0]) # such type of indexing is not applicable in dictionary

print(marks.keys())
```

```
print(marks)
marks["physics"] = 90
print(marks)
print(marks.items())
```

```
81
dict_keys(['physics', 'chemistry', 'maths', 'english'])
{'physics': 81, 'chemistry': 85, 'maths': 78, 'english': 81}
{'physics': 90, 'chemistry': 85, 'maths': 78, 'english': 81}
dict_items([('physics', 90), ('chemistry', 85), ('maths', 78), ('english', 81)])
```

In [218…
```
for k,v in marks.items():
    print(f"key is {k} and value is {v}")
```

```
key is physics and value is 90
key is chemistry and value is 85
key is maths and value is 78
key is english and value is 81
```

Methods on dictionary:

- clear() #Removes all the elements from the dictionary
- copy() #Returns a copy of the dictionary
- fromkeys() #Returns a dictionary with the specified keys and value
- get() #Returns the value of the specified key
- items() #Returns a list containing a tuple for each key value pair
- keys() #Returns a list containing the dictionary's keys
- pop() #Removes the element with the specified key
- popitem() #Removes the last inserted key-value pair
- setdefault() #Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
- update() #Updates the dictionary with the specified key-value pairs
- values() #Returns a list of all the values in the dictionary

Functions in python:

- In-built functions
- User-defined functions
- Module functions

In-built functions

In [128…
```
print("I'm an in-built function!")
int(4.5)
```

```
I'm an in-built function!
```
Out[128]:
```
4
```

Module functions

In [221…
```
#import math
#math.sqrt(22)
from math import sqrt as square_root
square_root(27)
```

Out[221]:
```
5.196152422706632
```

In [131…
```
from math import pow
pow(3,4)
```

Out[131]:
```
81.0
```

In [135…
```
# to see all the function defined in a module, use dir(module_name)
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos', 'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose', 'isfinite', 'isinf', 'isnan', 'isqrt', 'lcm', 'ldexp', 'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'nan', 'nextafter', 'perm', 'pi', 'pow', 'prod', 'radians', 'remainder', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc', 'ulp']
```

User-defined functions

Syntax:

```
def function_name(parameters):
    function body
    return something # default return value is null
```

Note: Function should be defined (and a module should be imported) before it is being used (called).

In [136…
```
def add(a,b):
    return a+b

print(add(10,12))
```

```
22
```

In [222…
```
def say_hello(name):
    print(f"Hello {name}!")
say_hello("Nitin")
print(say_hello("Thokare"))
```

```
Hello Nitin!
Hello Thokare!
None
```

Classes and objects

- Class is the most fundamental piece of Python. The reason lays behind the concept of object oriented programming.
- Everything in Python is an object such as integers, lists, dictionaries, functions and so on. Every object has a type and the object types are created using classes.
- Classes have:
  - Data attributes: Define what is needed to create an instance of a class
  - Methods (i.e. procedural attributes): Define how to interact with the instances of a class
  - Methods are just like functions but they belong to a particular class. The attributes can be considered as an interface to interact with a class.

- An advantage of classes is that we do not need to know how it is created. We can just use it through data attributes and methods.

Syntax of class definition:

```
class class_name(parent_class):
    def __init__(self, parameters):
        # attributes
        # attribute initialisation
    # class methods
```

In [201…

```
class Person():
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def get_name(self):
        return self.name
    def update_name(self,new_name): # by default it returns null (None)
        self.name = new_name

p = Person("Amish Sinha", 35)

print(p.get_name())
print(p.update_name("Amisha Sinha"))
print(p.get_name())
```

```
Amish Sinha
None
Amisha Sinha
```

Class Inheritance:

In [223…

```
class Person():
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def get_name(self):
        return self.name
    def update_name(self,new_name):
        self.name = new_name

class Student(Person):
    def __init__(self, name, age, roll_number):
        super().__init__(name, age)
        self.roll_number = roll_number

    def get_roll_number(self):
        return self.roll_number

p = Person("Amish Sinha", 35)
s = Student("Sumit Singh", 16, 21003)

print(p.get_name())
print(p.update_name("Amisha Sinha"))
print(p.get_name())
# print(p.get_roll_number()) # error: 'Person' object has no attribute 'get_roll_number'

print(s.get_name())
print(s.get_roll_number())
```

Amish Sinha
None
Amisha Sinha
Sumit Singh
21003