Agenda

- ❏ Introduction
- ❏ Development environment
- ❏ Basic data types
- ❏ Data structures
- ❏ Loops
- ❏ Conditions
- ❏ Functions
- ❏ OOPS
- ❏ Ways of programming
- ❏ File IO operations
- ❏ Packaging

Introduction

- Scripting language
- 2 versions of python are Python 2 and Python 3
- Unlike other programming languages, python is more dynamic so it is easy to learn.
- In 2 ways we can do scripting
  - Functional way
  - Object Oriented way
- It can wrap libraries written in other languages like C, CPP so it makes easy to port or enhance existing applications.
- Open source and community contribution is more.
- Most popular in different types of industries like automotive, web applications, streaming application. (YouTube, Facebook uses python for it’s purpose. Lot of AI based frameworks supports python like scikit, TensorFlow, pytorch)

Development environment

Python IDLE, Pycharm, eclipse (pydev plugin), Jupyter notebook (browser based on local/cloud)

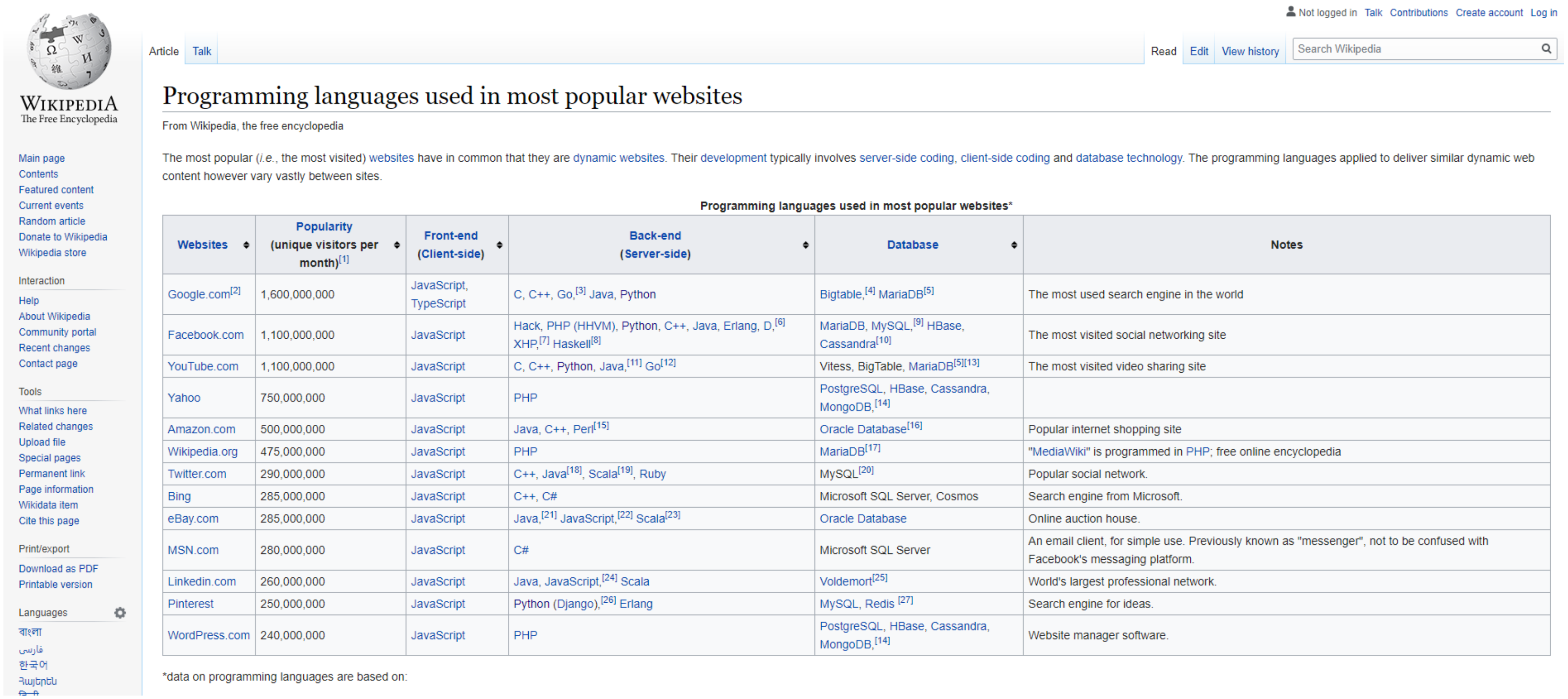
- **Pycharm**
  - Best for python scripting, best suits for application development
  - Project view makes easy to debug
  - Lots of shortcuts to make development easy
  - Git integration
  - Virtual environment can be created for specific project (much easy)
  - Pylint /pycode can be integrated
  - Unitest environment/template can be created and executed easily
  - Anaconda can be integrated (scientific package manager)
  - ….. Lot and lot features

  Refer for installation: https://www.jetbrains.com/help/pycharm/installation-guide.html

- **Jupyter notebook**
  - Useful during learning time
  - Especially helpful for teaching
  - Ploting using matplotlib can be done in same window

  Install: *pip install jupyterlab*

Companies using Python



Programming languages used in most popular websites

From Wikipedia, the free encyclopedia

The most popular (*i.e.*, the most visited) websites have in common that they are dynamic websites. Their development typically involves server-side coding, client-side coding and database technology. The programming languages applied to deliver similar dynamic web content however vary vastly between sites.

Programming languages used in most popular websites*

Websites	Popularity (unique visitors per month)[1]	Front-end (Client-side)	Back-end (Server-side)	Database	Notes
Google.com[2]	1,600,000,000	JavaScript, TypeScript	C, C++, Go,[3] Java, Python	Bigtable,[4] MariaDB[5]	The most used search engine in the world
Facebook.com	1,100,000,000	JavaScript	Hack, PHP (HHVM), Python, C++, Java, Erlang, D,[6] XHP,[7] Haskell[8]	MariaDB, MySQL,[9] HBase, Cassandra[10]	The most visited social networking site
YouTube.com	1,100,000,000	JavaScript	C, C++, Python, Java,[11] Go[12]	Vitess, BigTable, MariaDB[5][13]	The most visited video sharing site
Yahoo	750,000,000	JavaScript	PHP	PostgreSQL, HBase, Cassandra, MongoDB,[14]	
Amazon.com	500,000,000	JavaScript	Java, C++, Perl[15]	Oracle Database[16]	Popular internet shopping site
Wikipedia.org	475,000,000	JavaScript	PHP	MariaDB[17]	"MediaWiki" is programmed in PHP; free online encyclopedia
Twitter.com	290,000,000	JavaScript	C++, Java[18], Scala[19], Ruby	MySQL[20]	Popular social network.
Bing	285,000,000	JavaScript	C++, C#	Microsoft SQL Server, Cosmos	Search engine from Microsoft.
eBay.com	285,000,000	JavaScript	Java,[21] JavaScript,[22] Scala[23]	Oracle Database	Online auction house.
MSN.com	280,000,000	JavaScript	C#	Microsoft SQL Server	An email client, for simple use. Previously known as "messenger", not to be confused with Facebook's messaging platform.
Linkedin.com	260,000,000	JavaScript	Java, JavaScript,[24] Scala	Voldemort[25]	World's largest professional network.
Pinterest	250,000,000	JavaScript	Python (Django),[26] Erlang	MySQL, Redis [27]	Search engine for ideas.
WordPress.com	240,000,000	JavaScript	PHP	PostgreSQL, HBase, Cassandra, MongoDB,[14]	Website manager software.

*data on programming languages are based on:

Basic data types

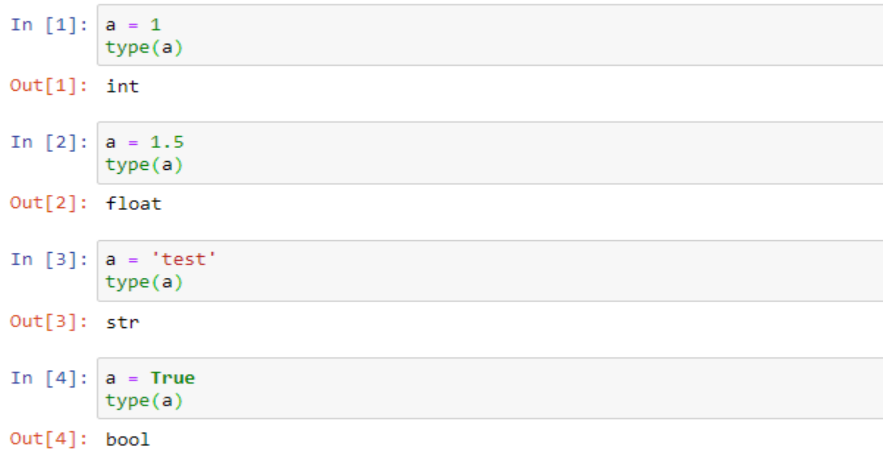
- Dynamic nature of python make automatic declaration of variables.

Example:

In C, to store data we need to declare variable at beginning. eg: int a, b.

In Python, a=2 automatically makes “a” as int type variable.

- Basic data types in python are **int, float, string, Boolean**.



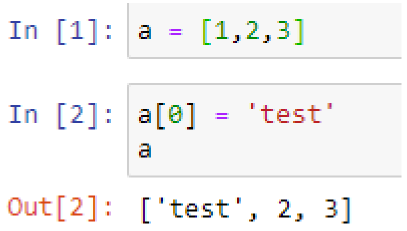
```
In [1]:  a = 1
         type(a)
Out[1]:  int

In [2]:  a = 1.5
         type(a)
Out[2]:  float

In [3]:  a = 'test'
         type(a)
Out[3]:  str

In [4]:  a = True
         type(a)
Out[4]:  bool
```

Data structure

Composite/abstract data types

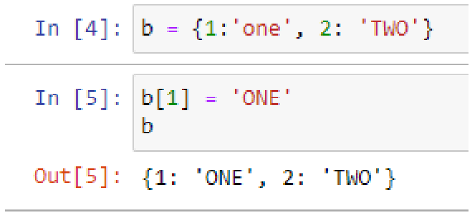
- **Mutable –** list, dictionary

list



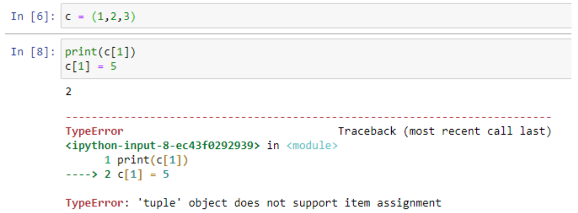
```
In [1]: a = [1,2,3]

In [2]: a[0] = 'test'
        a

Out[2]: ['test', 2, 3]
```

dictionary

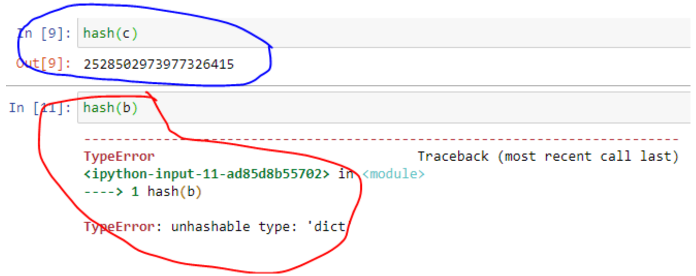


```
In [4]: b = {1:'one', 2: 'TWO'}

In [5]: b[1] = 'ONE'
        b

Out[5]: {1: 'ONE', 2: 'TWO'}
```

- **Immutable (** hashable) –** tuple, string, int, float, bool



```
In [6]: c = (1,2,3)

In [8]: print(c[1])
        c[1] = 5

2

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-ec43f0292939> in <module>
      1 print(c[1])
----> 2 c[1] = 5

TypeError: 'tuple' object does not support item assignment
```

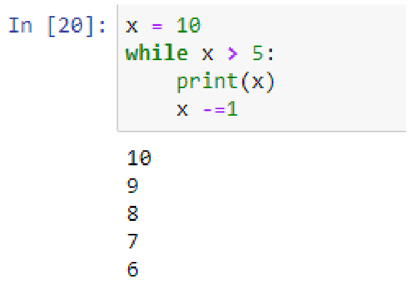


```
In [9]: hash(c)

Out[9]: 2528502973977326415

In [11]: hash(b)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-ad85d8b55702> in <module>
----> 1 hash(b)

TypeError: unhashable type: 'dict'
```

***Quick search – Especially useful in machine learning/text processing*

Loops

- while

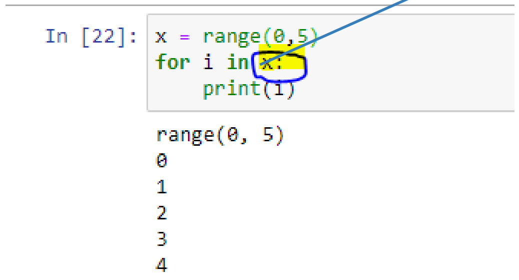


```
In [20]: x = 10
         while x > 5:
             print(x)
             x -=1
```

```
10
9
8
7
6
```

- for

X should be iterable. eg. list

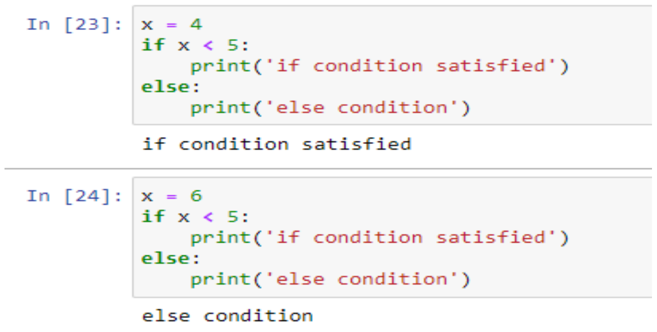


```
In [22]: x = range(0,5)
         for i in x:
             print(i)
```

```
range(0, 5)
0
1
2
3
4
```

Conditions

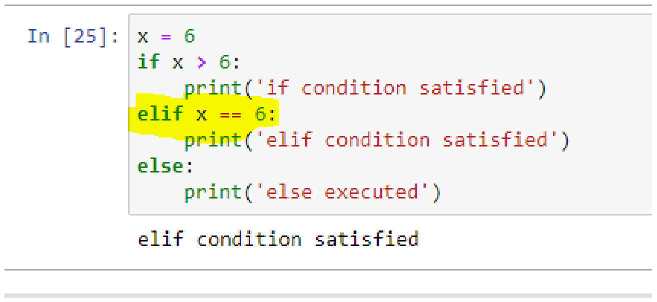
- if .. else



```
In [23]: x = 4
         if x < 5:
             print('if condition satisfied')
         else:
             print('else condition')

         if condition satisfied
```

```
In [24]: x = 6
         if x < 5:
             print('if condition satisfied')
         else:
             print('else condition')

         else condition
```

- if .. elif ..else

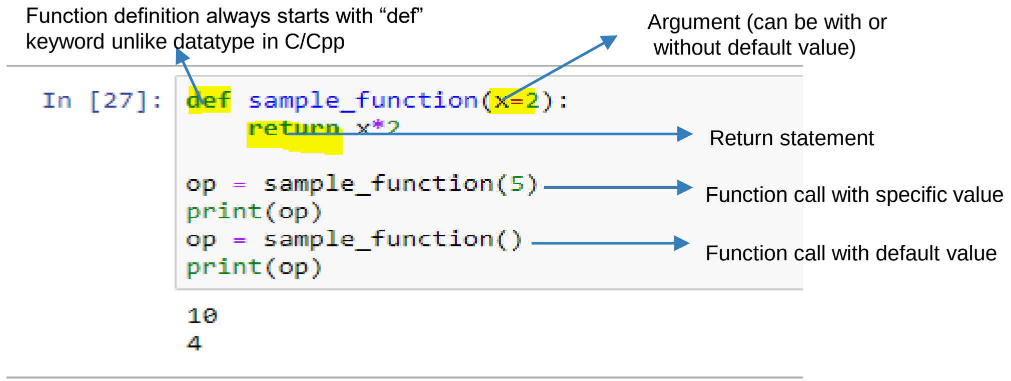


```
In [25]: x = 6
         if x > 6:
             print('if condition satisfied')
         elif x == 6:
             print('elif condition satisfied')
         else:
             print('else executed')

         elif condition satisfied
```

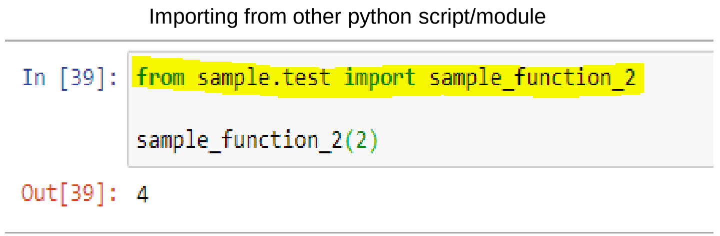
Note: *switch case not available in python*

Functions

- It is a small code, to do a specific task.
- It shall be called by anyone in same script and also can be imported/used in other python script.
- It may or may not take inputs (arguments).
- There is no need to tell return type like in C/CPP.


Function definition always starts with “def” keyword unlike datatype in C/Cpp
Argument (can be with or without default value)
Return statement
Function call with specific value
Function call with default value

```
In [27]: def sample_function(x=2):
             return x*2

         op = sample_function(5)
         print(op)
         op = sample_function()
         print(op)

         10
         4
```



```
In [39]: from sample.test import sample_function_2

         sample_function_2(2)

Out[39]: 4
```

Note:
Package is collection of modules.
Module is one python script with one or several function/class

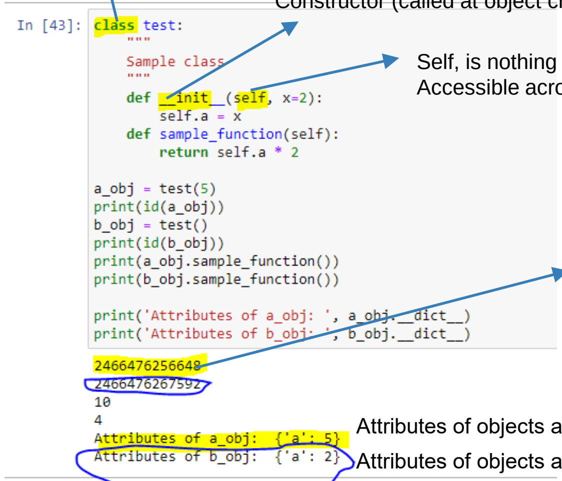
OOP – Object Oriented Programming

- **Object** is blueprint of class.
- **Class** is collection of attributes and methods.
- Separate space for each object so it is unique from other objects of same class.

Function definition always starts with “class”

Constructor (called at object creation time)

Self, is nothing but instance (only specific to that object).
Accessible across all methods of same object



```
In [43]: class test:
             """
             Sample class
             """
             def __init__(self, x=2):
                 self.a = x
             def sample_function(self):
                 return self.a * 2

         a_obj = test(5)
         print(id(a_obj))
         b_obj = test()
         print(id(b_obj))
         print(a_obj.sample_function())
         print(b_obj.sample_function())

         print('Attributes of a_obj: ', a_obj.__dict__)
         print('Attributes of b_obj: ', b_obj.__dict__)

         2466476256648
         2466476267592
         10
         4
         Attributes of a_obj:  {'a': 5}
         Attributes of b_obj:  {'a': 2}
```

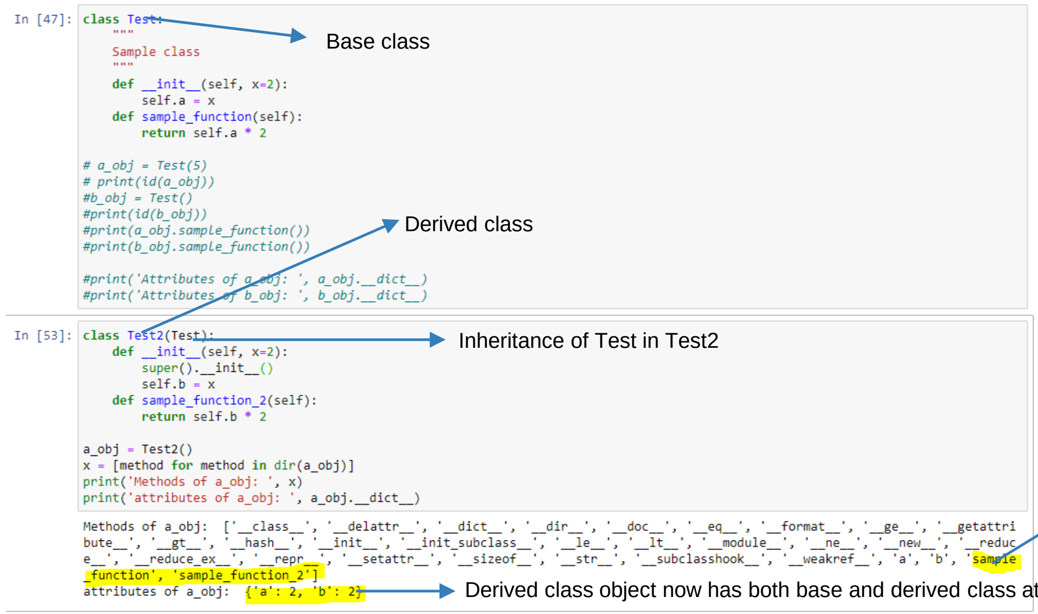
Object id (unique from other object)

Attributes of objects and it’s value

Attributes of objects and it’s value

OOP

Inheritance – Inheriting attributes and methods of base class



```
In [47]: class Test:
             """
             Sample class
             """
             def __init__(self, x=2):
                 self.a = x
             def sample_function(self):
                 return self.a * 2

         # a_obj = Test(5)
         # print(id(a_obj))
         #b_obj = Test()
         #print(id(b_obj))
         #print(a_obj.sample_function())
         #print(b_obj.sample_function())

         #print('Attributes of a_obj: ', a_obj.__dict__)
         #print('Attributes of b_obj: ', b_obj.__dict__)
```

Base class

Derived class

```
In [53]: class Test2(Test):
             def __init__(self, x=2):
                 super().__init__()
                 self.b = x
             def sample_function_2(self):
                 return self.b * 2

         a_obj = Test2()
         x = [method for method in dir(a_obj)]
         print('Methods of a_obj: ', x)
         print('attributes of a_obj: ', a_obj.__dict__)

         Methods of a_obj:  ['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattri
         bute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduc
         e__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'a', 'b', 'sample
         _function', 'sample_function_2']
         attributes of a_obj:  {'a': 2, 'b': 2}
```

Inheritance of Test in Test2

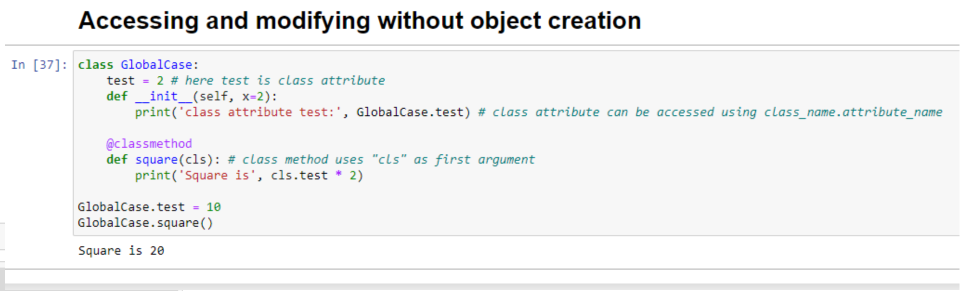
Derived class object now has both base and derived class methods

Derived class object now has both base and derived class attributes

OOP

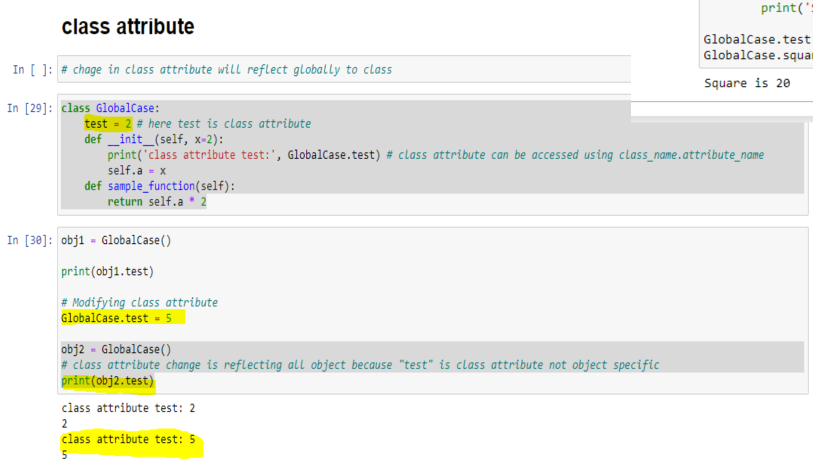
Class attribute and class method

- Attributes and methods can be accessed and modified without creating object.
- Changes will be reflected in all object that accessing class attributes



Accessing and modifying without object creation

```
In [37]: class GlobalCase:
             test = 2 # here test is class attribute
             def __init__(self, x=2):
                 print('class attribute test:', GlobalCase.test) # class attribute can be accessed using class_name.attribute_name

             @classmethod
             def square(cls): # class method uses "cls" as first argument
                 print('Square is', cls.test * 2)

         GlobalCase.test = 10
         GlobalCase.square()

         Square is 20
```



class attribute

```
In [ ]: # chage in class attribute will reflect globally to class

In [29]: class GlobalCase:
             test = 2 # here test is class attribute
             def __init__(self, x=2):
                 print('class attribute test:', GlobalCase.test) # class attribute can be accessed using class_name.attribute_name
                 self.a = x
             def sample_function(self):
                 return self.a * 2

In [30]: obj1 = GlobalCase()

         print(obj1.test)

         # Modifying class attribute
         GlobalCase.test = 5

         obj2 = GlobalCase()
         # class attribute change is reflecting all object because "test" is class attribute not object specific
         print(obj2.test)

         class attribute test: 2
         2
         class attribute test: 5
         5
```