Learn python for Django

Just Enough python to get into web development!

By Samir Phuyal

Data Types in Python

< SP />

Data Types and Structure in Python

STRINGS : ‘Hello pythonista ‘ , “Earth is green”, ‘computers’

INTEGERS: 100, 365, 786

FLOATS: 12.6, 10.0, 3790.12

BOOLEANS: True, False

LISTS: [100, ”computers”, 12.6, ”Earth is green”]

DICTIONARIES: { “name”:”pythonista”, “number”:1, “height”:”5.5 ft” }

TUPLES: (100, ”computers”, 12.6, “Earth is green”)

SETS: { “python”, “C”, “Java”, “JavaScript” }

NOTE:- I will show you each data types in more detail on using it in coming videos.

Variables, Operators & Inputs

< SP />

What is a variable?

- Stores data or value
- Needs to be assigned
- Can store anything
- Can be called only after assigning it

E.g:-

```
name = “Pythonista”
```

‘ = ‘ Tells python to assign Pythonista string to name

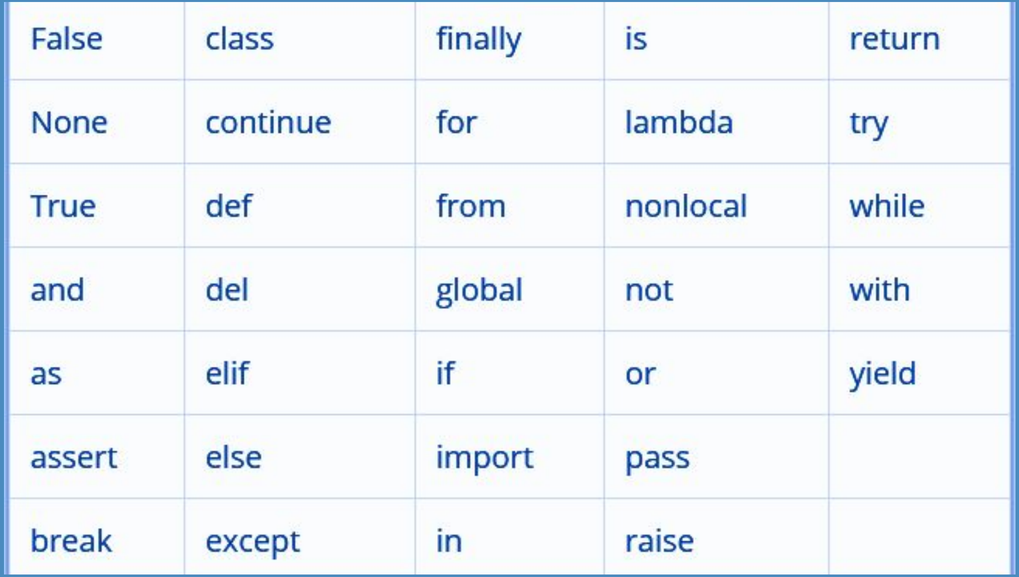
* Here name is a variable and pythonista is a stored string value

Conventions of naming variables

- Make it descriptive / easy to read
- Constant values can be uppercase
- Should not begin with numbers
- Words can be separated by underscore
- Don’t use python reserved keywords

* I will give you example of it throughout the series

Python Keywords

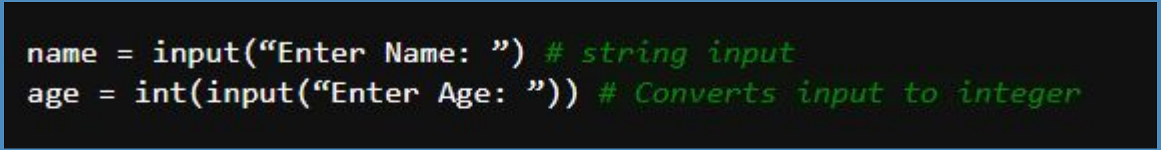


False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

OPERATORS

- Arithmetic operators +, -, *, /,% , //, **
- Comparison operators >, <, ==, !=, <=, >=
- Assignment operators =, +=, -=, *=, //=, %=, **=
- Membership & Identity operators (and,is,not,or,in)

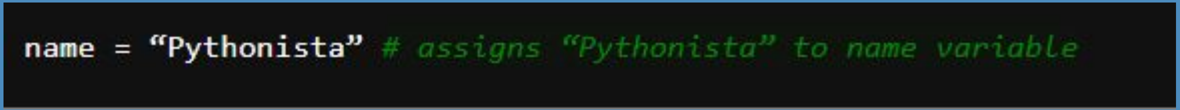
INPUT



```
name = input("Enter Name: ") # string input
age = int(input("Enter Age: ")) # Converts input to integer
```

- Uses input()
- Can be stored in variable: name = input("Enter Name: ")
- Takes string values by default
- Integer input by: age = int(input("Enter Age: "))
- Throws errors when user give unexpected input data

COMMENTS



```
name = “Pythonista” # assigns “Pythonista” to name variable
```

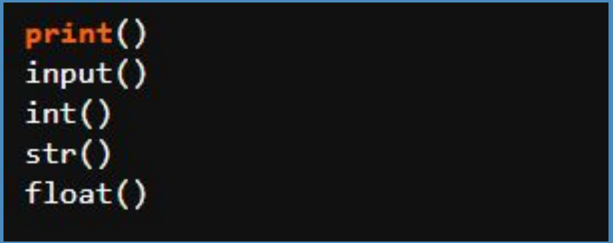
Here, anything after # will not be executed but it explains what that line of code is doing by writing a comment.

- Starts with #
- Whatever comes after # is ignored by computer
- Use to explain what your code is doing

Built-in Functions & Methods

< SP />

Built-in Functions

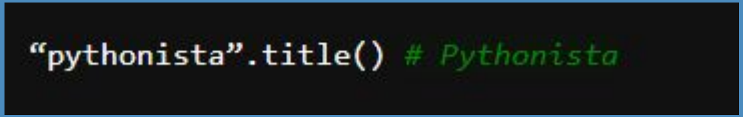


```
print()
input()
int()
str()
float()
```

- Special Functionality in python
  - Syntax is like name()
- Already defined in python

Built-in Methods

- Associated to end of specific data types
- Syntax : value.method()
- Does something to that value
- Strings, Lists, Dictionaries all have their own methods



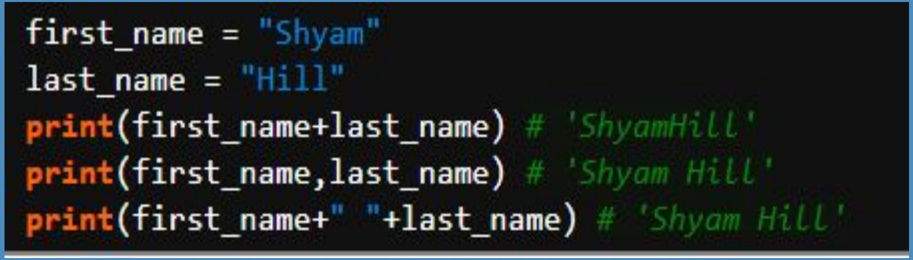
```
“pythonista”.title() # Pythonista
```

* They will be discussed on seperate videos on more detail.

STRINGS IN PYTHON

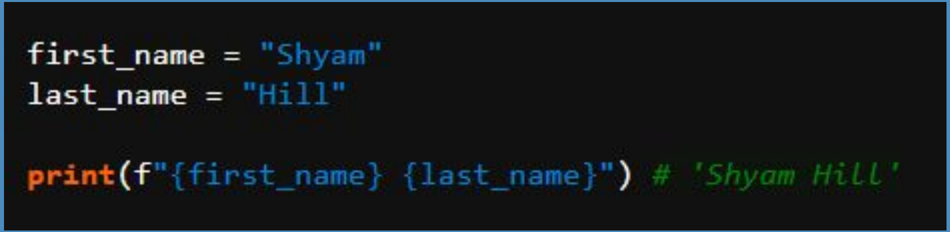
< SP />

Adding Strings by Concatenation


```
first_name = "Shyam"
last_name = "Hill"
print(first_name+last_name) # 'ShyamHill'
print(first_name,last_name) # 'Shyam Hill'
print(first_name+" "+last_name) # 'Shyam Hill'
```

* Note that python is case-sensitive
name is not equal to Name in python.

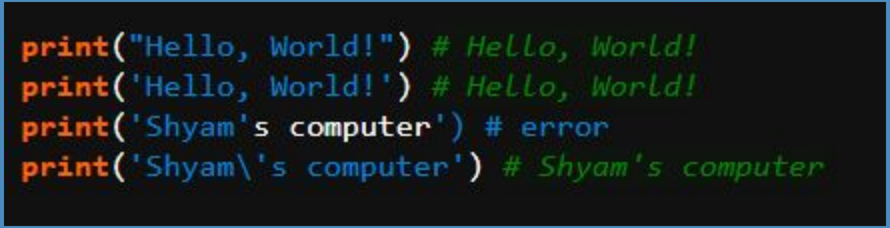
String Interpolation



```
first_name = "Shyam"
last_name = "Hill"

print(f"{first_name} {last_name}") # 'Shyam Hill'
```

* Note that variables must be inside { } and ‘ f ’ should be at beginning.

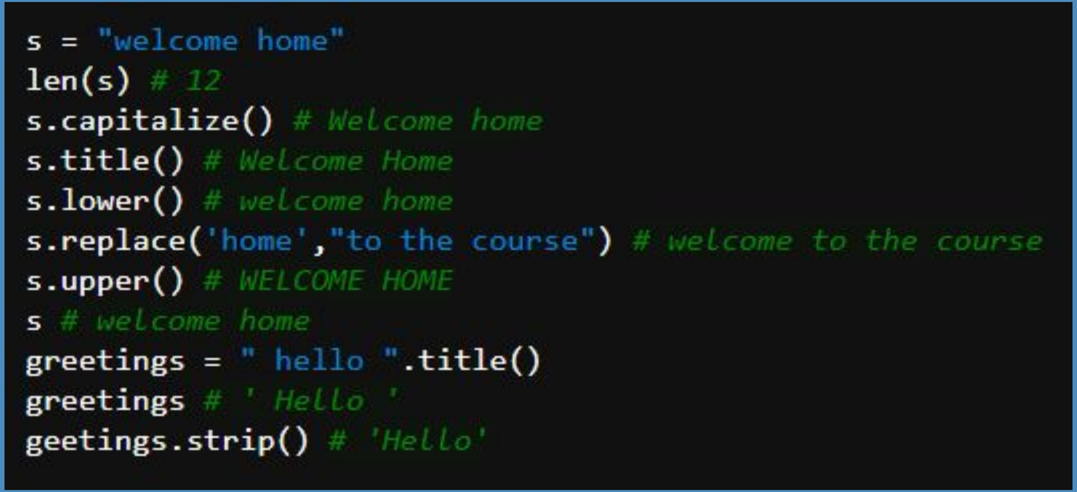
String Options:



```
print("Hello, World!") # Hello, World!
print('Hello, World!') # Hello, World!
print('Shyam's computer') # error
print('Shyam\'s computer') # Shyam's computer
```

* You can find more about escape characters in course reference guide too.

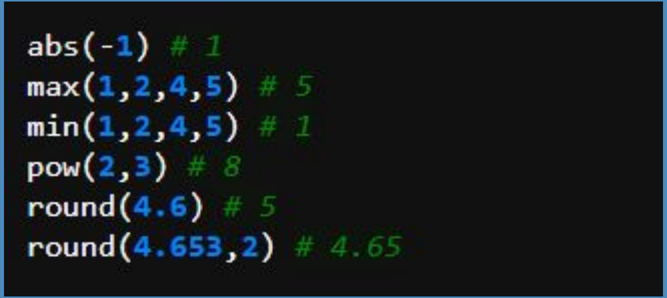
String Built-in Methods:


```
s = "welcome home"
len(s) # 12
s.capitalize() # Welcome home
s.title() # Welcome Home
s.lower() # welcome home
s.replace('home',"to the course") # welcome to the course
s.upper() # WELCOME HOME
s # welcome home
greetings = " hello ".title()
greetings # ' Hello '
geetings.strip() # 'Hello'
```

NUMBERS IN PYTHON

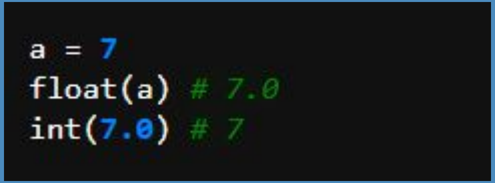
< SP />

Methods related to numbers



```
abs(-1) # 1
max(1,2,4,5) # 5
min(1,2,4,5) # 1
pow(2,3) # 8
round(4.6) # 5
round(4.653,2) # 4.65
```

Number type conversion



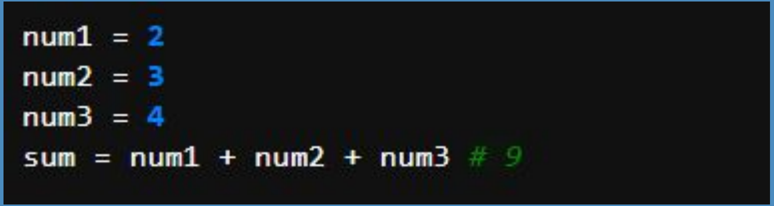
```
a = 7
float(a) # 7.0
int(7.0) # 7
```

LISTS IN PYTHON

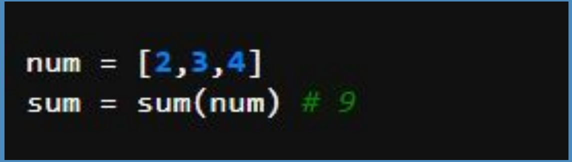
< SP />

What is list and why use it?

- Collection of values
- Each value separated by comma
- Simplifies data handling
- Can do large computation without more hassle

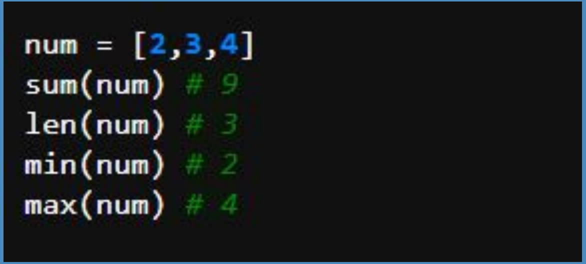


```
num1  = 2
num2  = 3
num3  = 4
sum  = num1 + num2 + num3 # 9
```



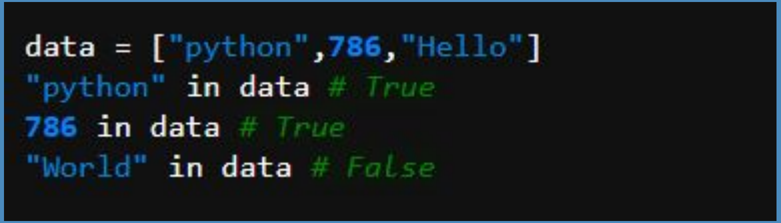
```
num  = [2,3,4]
sum  = sum(num) # 9
```

List Functions



```
num = [2,3,4]
sum(num) # 9
len(num) # 3
min(num) # 2
max(num) # 4
```

Check whether item is in list



```
data = ["python",786,"Hello"]
"python" in data # True
786 in data # True
"World" in data # False
```

How to access list values ?

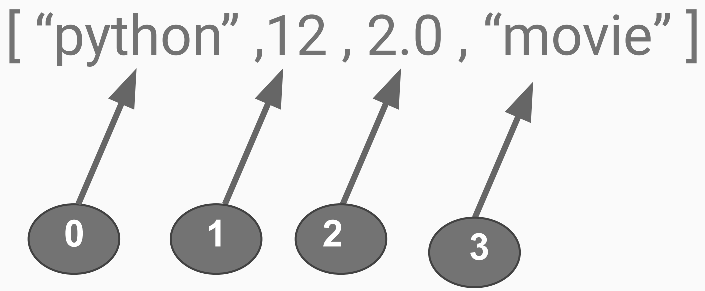
List indexes every values gradually and we can access those values by referring to their index number

Two ways of accessing list values:

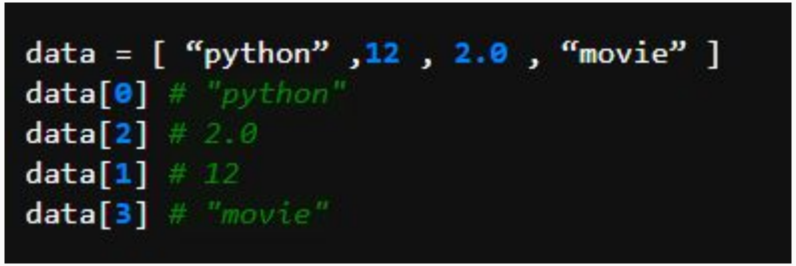
- Forward Indexing
- Backward Indexing

List[index]

FORWARD

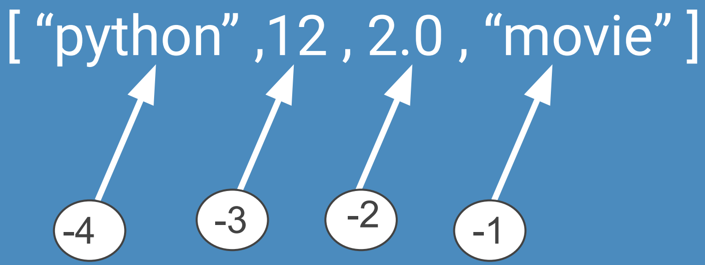


Starts with 0

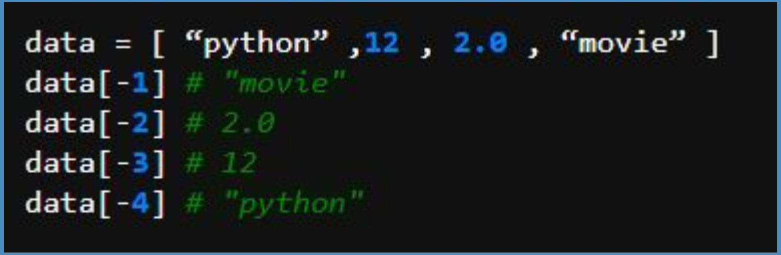


```
data = [ “python” ,12 , 2.0 , “movie” ]
data[0] # "python"
data[2] # 2.0
data[1] # 12
data[3] # "movie"
```

BACKWARD



Starts with - ve sign

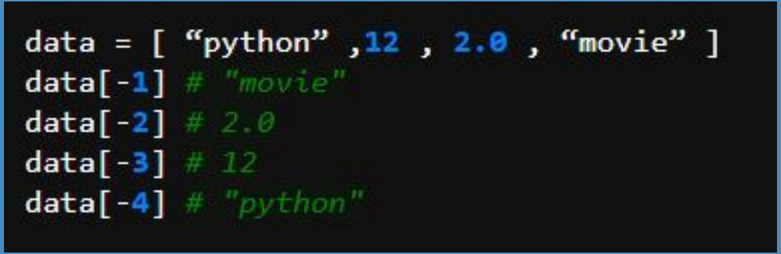


```
data = [ “python” ,12 , 2.0 , “movie” ]
data[-1] # "movie"
data[-2] # 2.0
data[-3] # 12
data[-4] # "python"
```

List Slicing to get sublist

[start : end]

1. Slices list including value of start index
2. Slices Upto end index but does not include end index value
3. Returns the sliced **list**

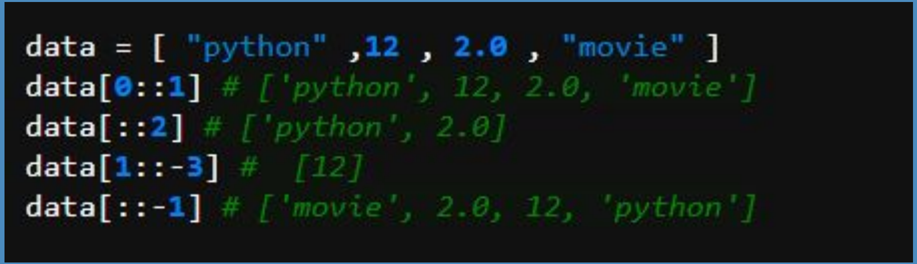


```
data = [ “python” ,12 , 2.0 , “movie” ]
data[-1] # "movie"
data[-2] # 2.0
data[-3] # 12
data[-4] # "python"
```

Third Slicing Parameter

[start : end : **step**]

1. Skips specific steps while slicing
2. Default step value is 1
3. Includes beginning index and skips the number of index in step



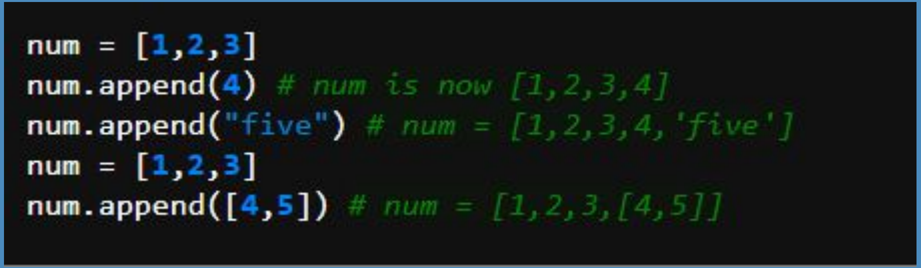
```
data = [ "python" ,12 , 2.0 , "movie" ]
data[0::1] # ['python', 12, 2.0, 'movie']
data[::2] # ['python', 2.0]
data[1::-3] #  [12]
data[::-1] # ['movie', 2.0, 12, 'python']
```

List Methods

1. append(object)
2. count(object)
3. extend(list)
4. index(object)
5. insert(index , object)
6. pop(object)
7. remove(object)
8. reverse()
9. sort()

APPEND

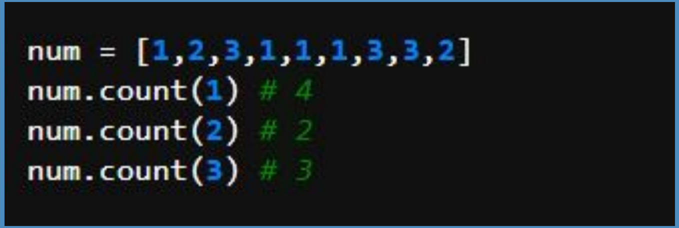
Adds an element to the end of the list



```
num = [1,2,3]
num.append(4) # num is now [1,2,3,4]
num.append("five") # num = [1,2,3,4,'five']
num = [1,2,3]
num.append([4,5]) # num = [1,2,3,[4,5]]
```

COUNT

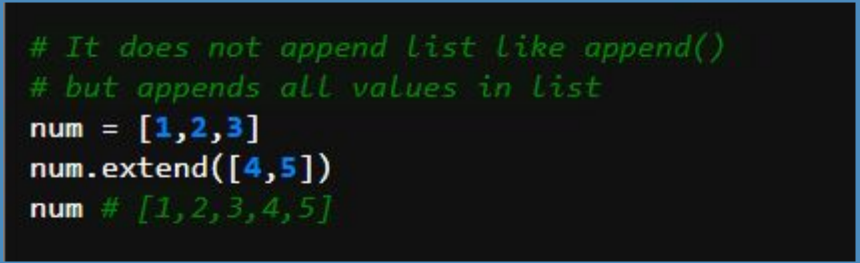
Returns the number of time a element is in the list.



```
num = [1,2,3,1,1,1,3,3,2]
num.count(1) # 4
num.count(2) # 2
num.count(3) # 3
```

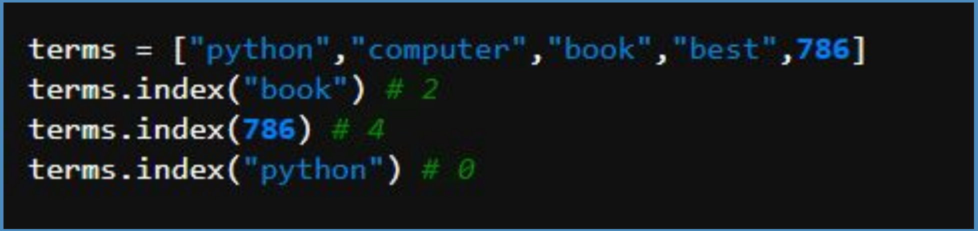
EXTEND

Appends all the elements in the list to the original list


```
# It does not append list like append()
# but appends all values in list
num = [1,2,3]
num.extend([4,5])
num # [1,2,3,4,5]
```

INDEX

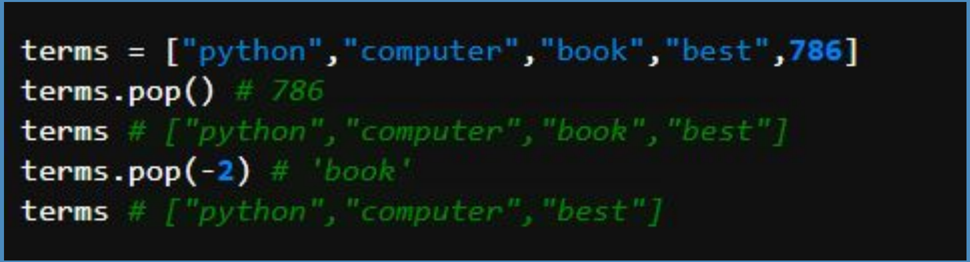
Returns the index number of the **first occurence** of element in the list.



```
terms = ["python","computer","book","best",786]
terms.index("book") # 2
terms.index(786) # 4
terms.index("python") # 0
```

POP

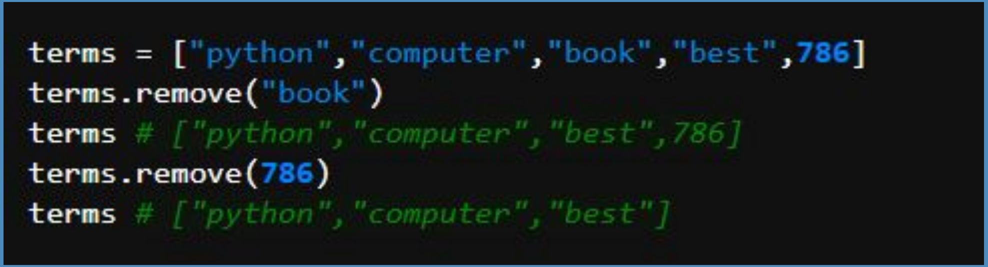
Removes the element at given index number and returns it.
If no parameter is passed then it removes last value of list and returns it.



```
terms = ["python","computer","book","best",786]
terms.pop() # 786
terms # ["python","computer","book","best"]
terms.pop(-2) # 'book'
terms # ["python","computer","best"]
```

REMOVE

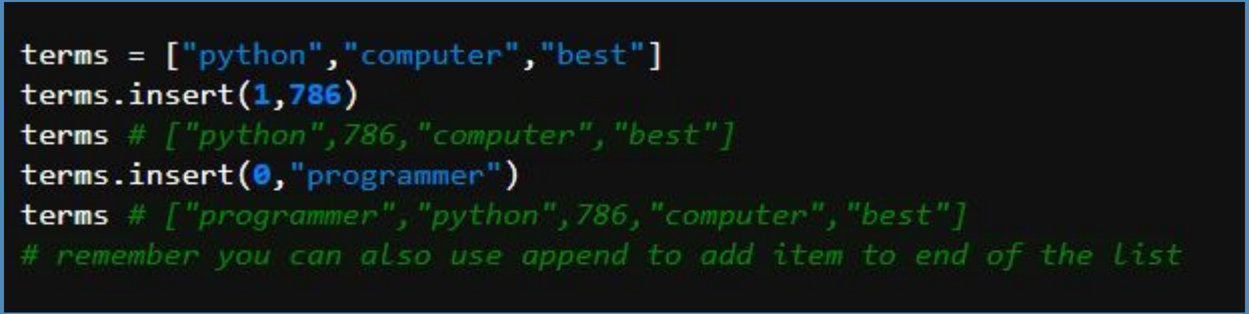
Removes the **first occurence** of the **passed element** from the list.



```
terms = ["python","computer","book","best",786]
terms.remove("book")
terms # ["python","computer","best",786]
terms.remove(786)
terms # ["python","computer","best"]
```

INSERT

Takes two parameters (x,object)
Inserts object at index x

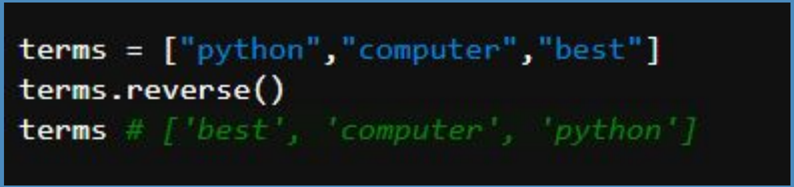


```
terms = ["python","computer","best"]
terms.insert(1,786)
terms # ["python",786,"computer","best"]
terms.insert(0,"programmer")
terms # ["programmer","python",786,"computer","best"]
# remember you can also use append to add item to end of the list
```

Hold on! Still 2 more methods left!

REVERSE

Reverses the elements in the list

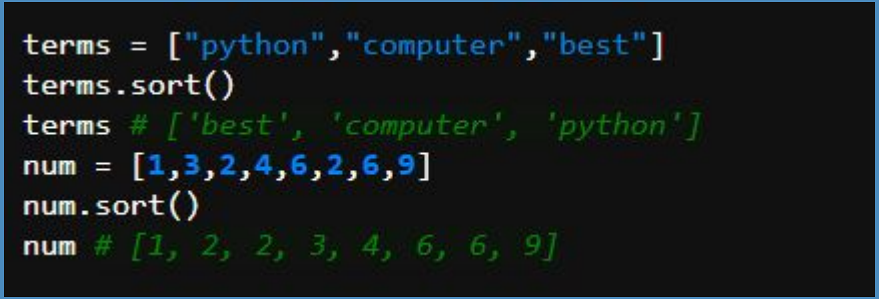


```
terms = ["python","computer","best"]
terms.reverse()
terms # ['best', 'computer', 'python']
```

Remember you can also use list slicing technique to reverse a list using [: : -1]

SORT

Sorts the elements in the list in ascending order.



```
terms = ["python","computer","best"]
terms.sort()
terms # ['best', 'computer', 'python']
num = [1,3,2,4,6,2,6,9]
num.sort()
num # [1, 2, 2, 3, 4, 6, 6, 9]
```

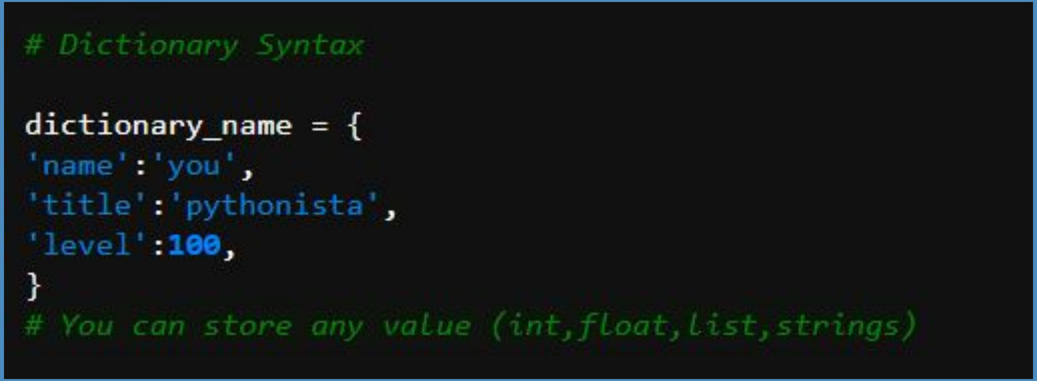
Remember you can also use list slicing technique to reverse a list using [: : -1]

DICTIONARIES

< SP />

What is dictionary and why even use it ?

Dictionary is a collection of variable and value or key and values
It helps to store detail information with informative **key**
You can **GET , EDIT and DELETE values using the key**



```
# Dictionary Syntax

dictionary_name = {
'name':'you',
'title':'pythonista',
'level':100,
}
# You can store any value (int,float,list,strings)
```

DICTIONARY OPERATIONS

ADDING VALUES: *dictionary_name[new_key] = value*

EDITING VALUES: *dictionary_name[key_to_change_value] = value*

DELETING VALUES: *del dictionary_name[key]*

Is Key in a DICTIONARY?

“key” in dictionary_name
# returns True or False whether key is in the dictionary or not !

DICTIONARY METHODS

KEYS() : *returns sequence of dictionary keys in tuple*

values() : *returns sequence of dictionary values in tuple*

items(): *returns sequence of (keys , values)*

clear(): *deletes all entries in dictionary*

get(key): *returns the value for the* ***KEY***

Tuples & Sets

< SP />

What is tuple and why even use it ?

Tuple are like lists but their elements are fixed.
You can't **ADD , EDIT , REPLACE , REORDER or DELETE elements in it.**
You can use it to prevent some data from being changed.

Syntax of tuple is like:

```
tuple_name = ( 1,2,3,4 )
```

Tuple uses small brackets not square brackets.
You can use len() , max(), min(), sum() functions in tuples also.
You can also slice tuples as same as in list

What are sets and why even use them ?

Sets are also like lists but their elements are not ordered in sequence.
If your application does not care about the order of data, then sets are helpful.

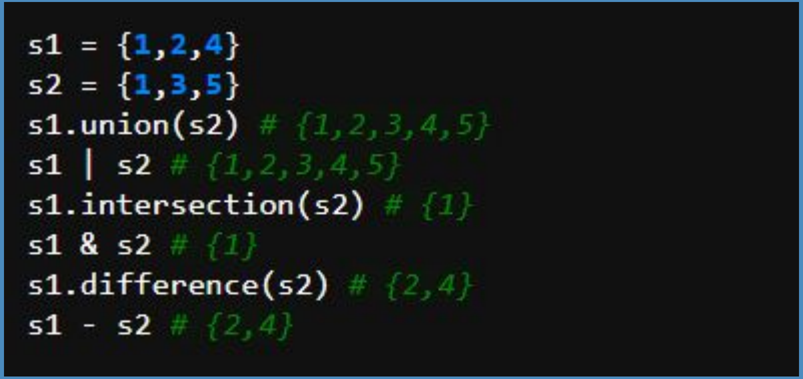
Syntax of set is like:

set_name = { 1,2,3,4 }

Set uses curly brackets not square brackets.
You can use len() , max(), min(), sum() functions in sets also.
You can't have duplicate item in tuple.

Set Operations

UNION : union() or ‘ | ‘
Intersection : intersection() or ‘ & ‘
Difference : difference() or ‘ - ‘



```
s1 = {1,2,4}
s2 = {1,3,5}
s1.union(s2) # {1,2,3,4,5}
s1 | s2 # {1,2,3,4,5}
s1.intersection(s2) # {1}
s1 & s2 # {1}
s1.difference(s2) # {2,4}
s1 - s2 # {2,4}
```

Conditionals - if with else

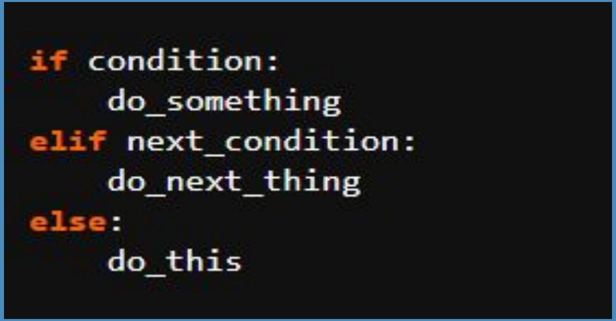
< SP />

Adding Logic to the code

- What to do if something happens?
- Runs specific code for specific condition
- Condition may be True or False

SYNTAX

SYNTAX



```
if condition:
    do_something
elif next_condition:
    do_next_thing
else:
    do_this
```

If condition in *if* statement does not matches then it checks in *elif* statement
And if nothing matches in *elif* too then code in *else* will run
*** elif is optional**

HOW TO KNOW TRUE or FALSE?

- Assignment operators =, +=, -=, *=, //=, %=, **=
- Using ‘ or ’ , ‘ and ’ , ‘ not ’ together
- Combine True or False for complex logic

Combining True with False

True or True => True
True or False => False
True and True => True
True and False => False
not True => False
not False => True

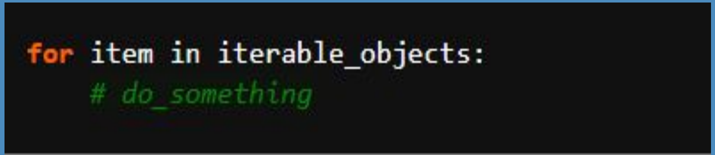
Loops - while & for

< SP />

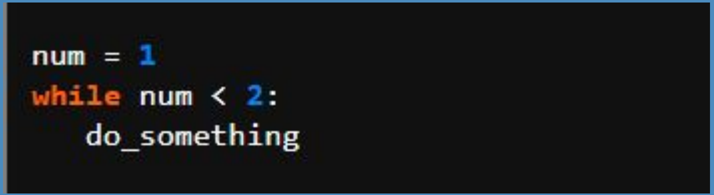
Why Loops?

- Helps to run specific code repeatedly
- Runs until certain condition is met
- Versatile - for and while loop

SYNTAX

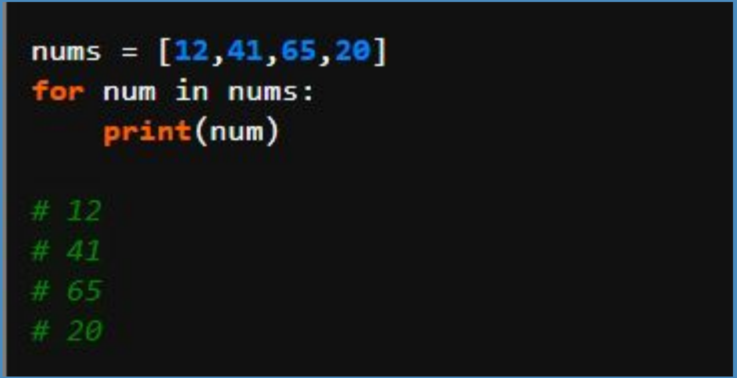


```
for item in iterable_objects:
    # do_something
```



```
num = 1
while num < 2:
    do_something
```

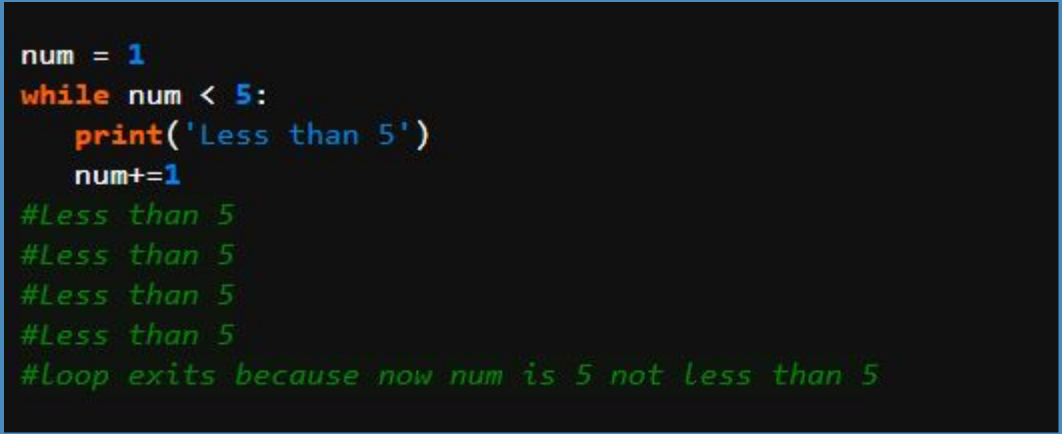
For Loop



```
nums = [12,41,65,20]
for num in nums:
    print(num)

# 12
# 41
# 65
# 20
```

Goes through each number in nums list
For each time num variable is assigned to that number
Next time that num variable is assigned to the next number in the list
We can add our code to do specific operation for each loop

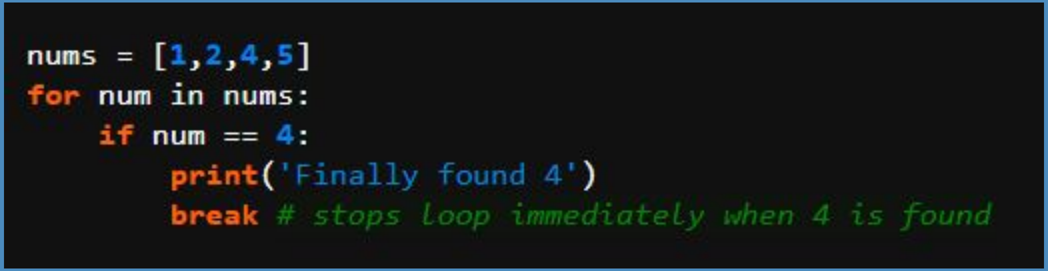
While Loop



```
num = 1
while num < 5:
    print('Less than 5')
    num+=1
#Less than 5
#Less than 5
#Less than 5
#Less than 5
#loop exits because now num is 5 not less than 5
```

Takes Given condition and if it is True then,
It will run its code else it will not
Remember if the condition is True always then,
Code will run forever so we here incremented num by 1 every time

Adding Logic with Loops



```
nums = [1,2,4,5]
for num in nums:
    if num == 4:
        print('Finally found 4')
        break # stops loop immediately when 4 is found
```

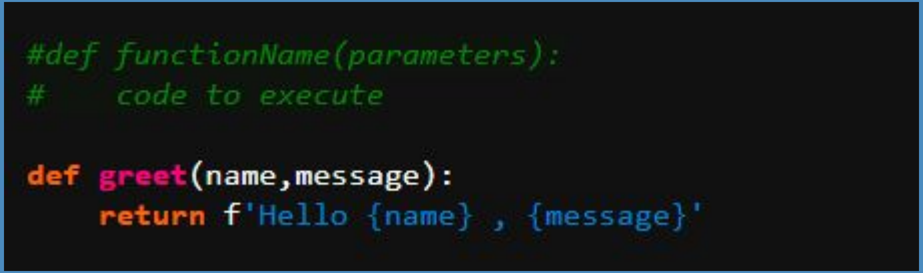
**Above code will return ‘Finally found 4’ when num is 4 in loop then ,
Exits from doing next loop because of break keyword**

Use break keyword to prevent infinite loop or to terminate loop.

Functions

< SP />

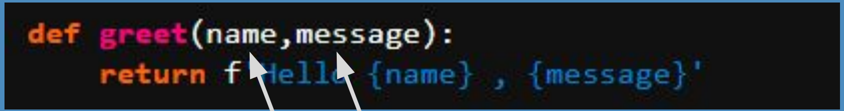
What is it & Why ?



```python
#def functionName(parameters):
#    code to execute

def greet(name,message):
    return f'Hello {name} , {message}'
```

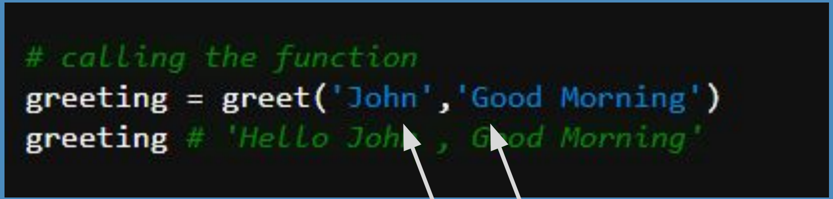
- Starts with def keyword
- Can take input as parameter
- Block of reusable code
- Needs to be defined before using it
- Can return the value based on parameters
- Can be used many time by calling it

Calling the function



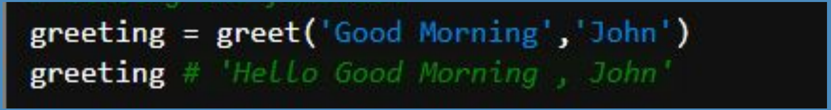
```python
def greet(name,message):
    return f'Hello {name} , {message}'
```

Parameters

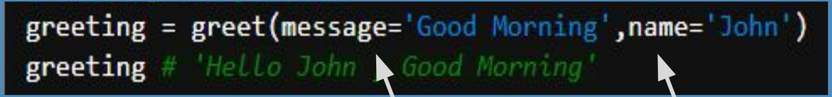


```python
# calling the function
greeting = greet('John','Good Morning')
greeting # 'Hello John , Good Morning'
```

Arguments



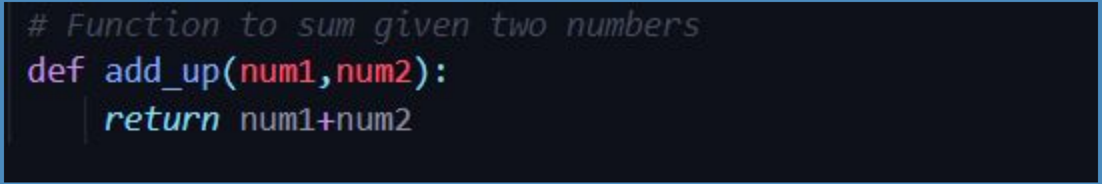
```python
greeting = greet('Good Morning','John')
greeting # 'Hello Good Morning , John'
```



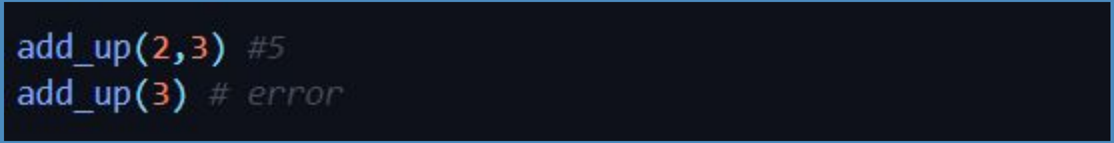
```python
greeting = greet(message='Good Morning',name='John')
greeting # 'Hello John , Good Morning'
```

Positional Arguments

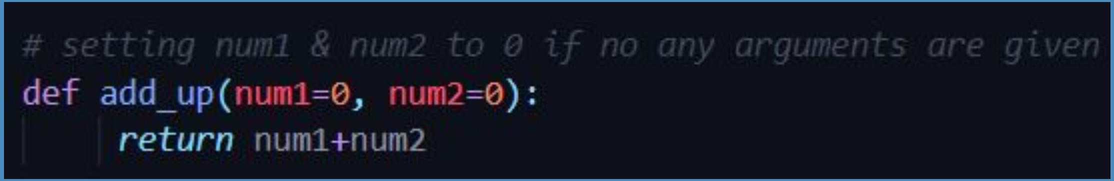
Passing default value in parameter



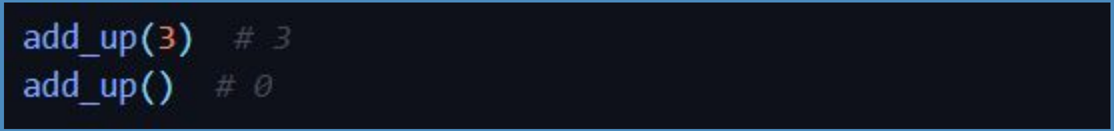
```python
# Function to sum given two numbers
def add_up(num1,num2):
    return num1+num2
```



```python
add_up(2,3) #5
add_up(3) # error
```



```python
# setting num1 & num2 to 0 if no any arguments are given
def add_up(num1=0, num2=0):
    return num1+num2
```



```python
add_up(3)  # 3
add_up()  # 0
```

Working with Functions

- Needs to return something to save output in a variable
- Can have not any parameters too
- Functions with parameters need to be called with arguments
- Arguments can be passed with parameter name for readability
- Can combine complex logic in the function
- Use it to run repetitive programming task

Object Oriented Programming

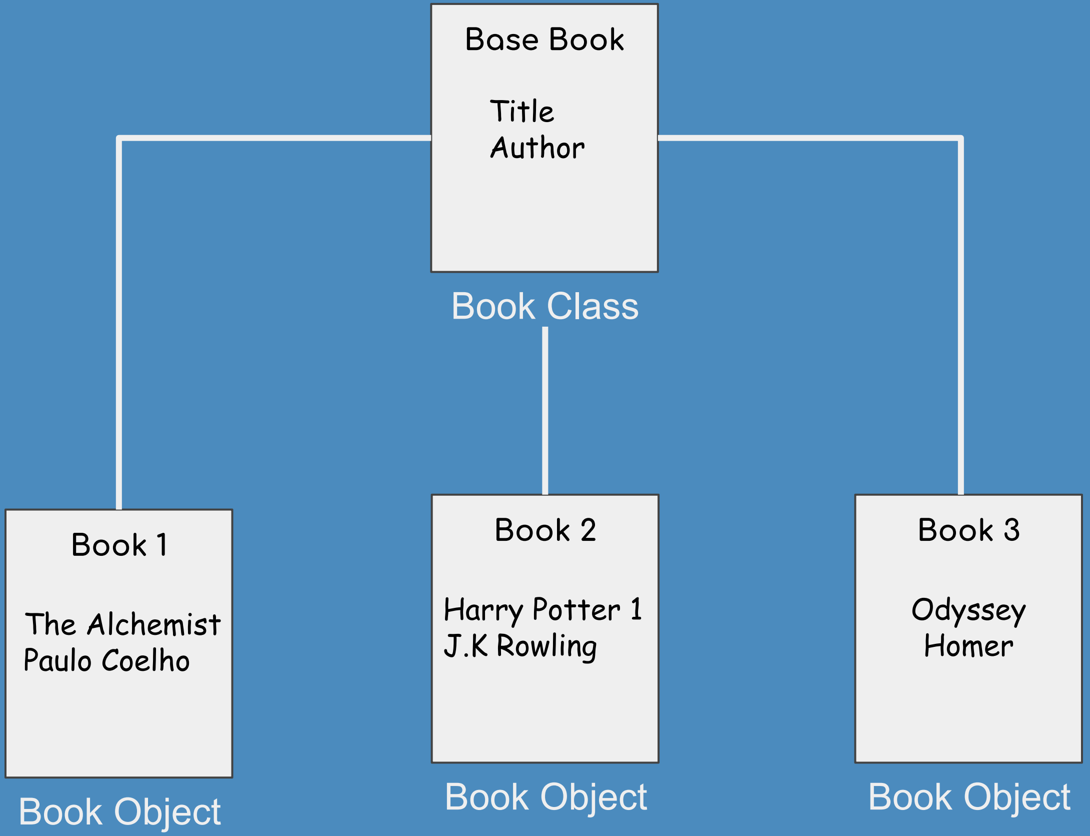
< SP />

What is OOP exactly?

OOP is the use of objects to create the program

- Object can be referred to anything - book,movie,toy etc
- Helps to assign specific functions,qualities,attributes in base class to create many objects from it effectively

How class and objects work?



SYNTAX

Creating class

class ClassName:
initializer
methods

Creating objects from class

object = **ClassName()**

Accessing object properties & methods

object.property
object.method()

Important Term

Initializer : special __init__ function to set variables or attributes to the object

Methods : functions defined in the class

What does __init__ do ?

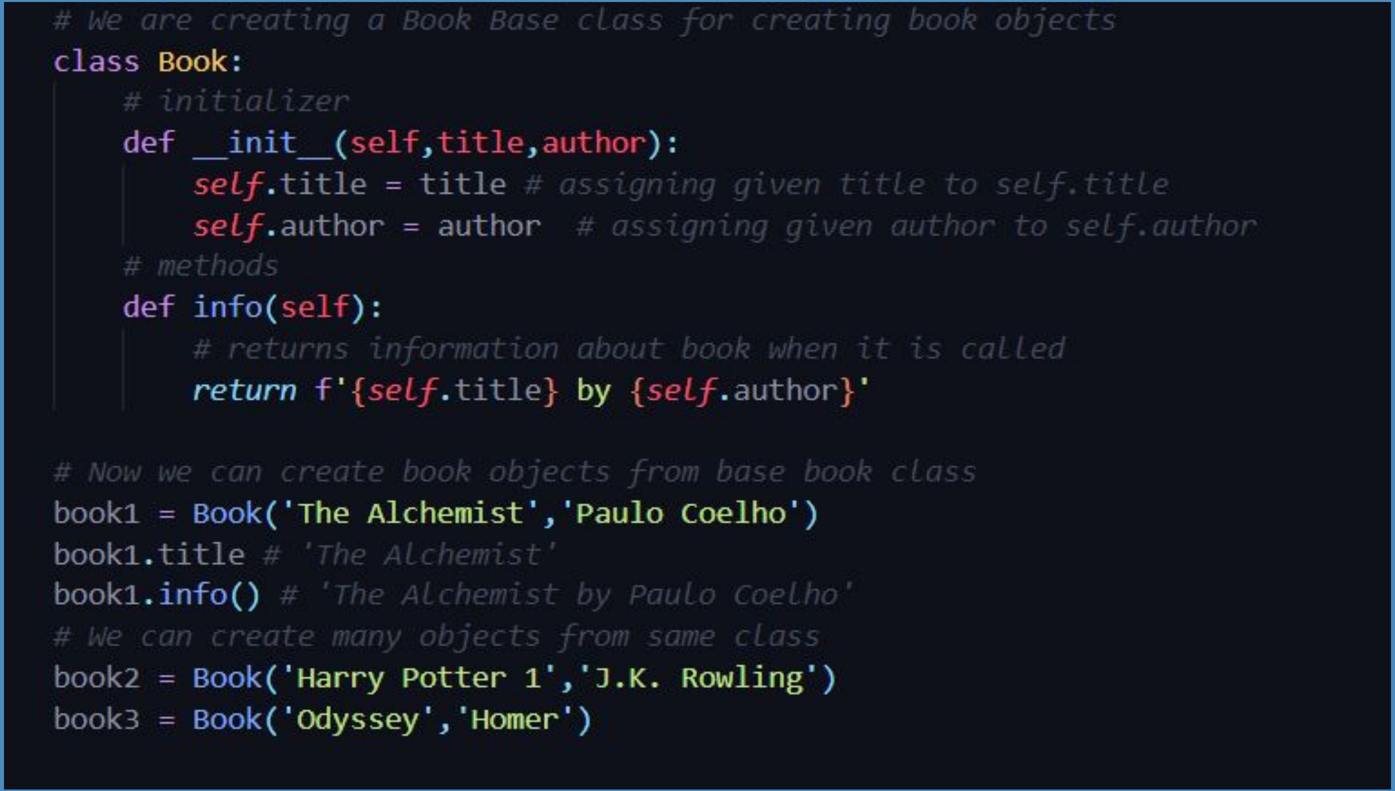
It is a special function that gets called automatically when new object is created from given data values

IMPORTANT NOTE

Every functions should have ‘ self ’ parameter in the class.

Every variable should also be accessed through ‘ self.variable ’

Creating book object from base class



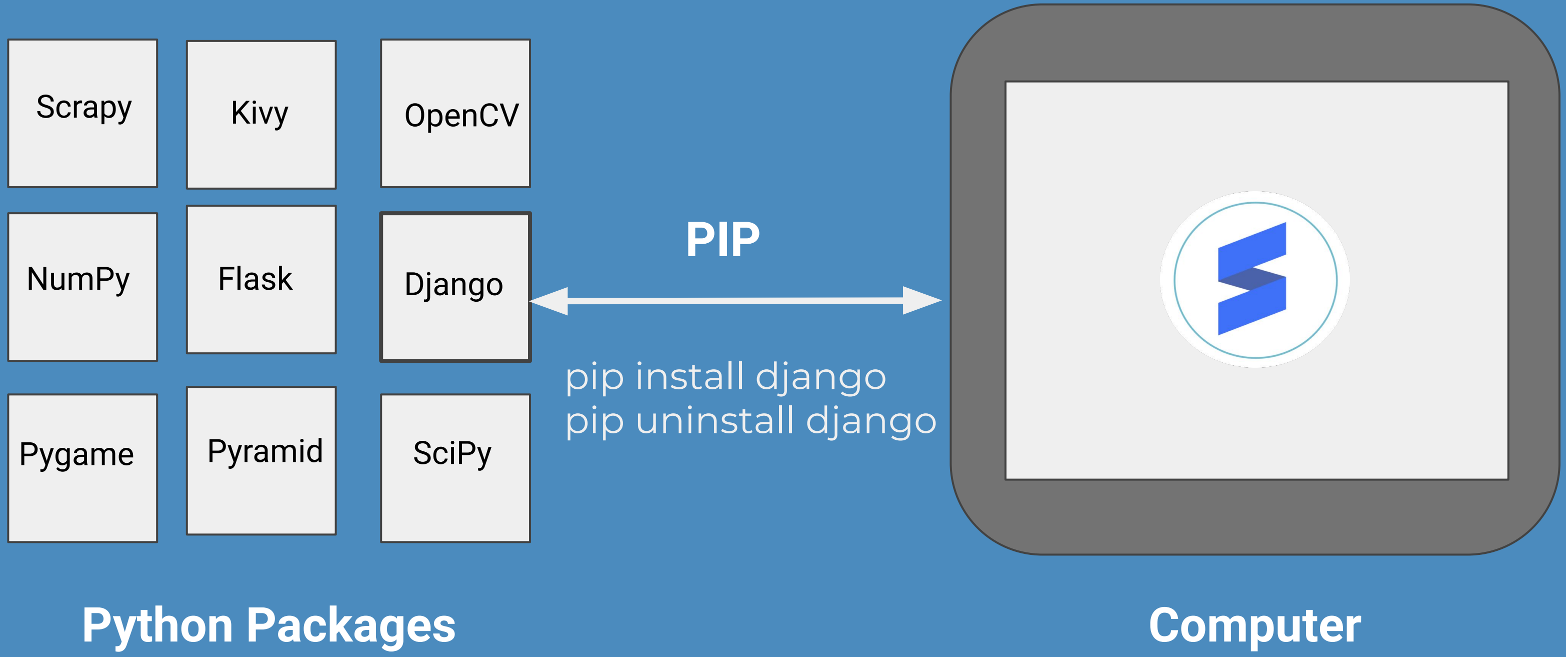
```python
# We are creating a Book Base class for creating book objects
class Book:
    # initializer
    def __init__(self,title,author):
        self.title = title # assigning given title to self.title
        self.author = author  # assigning given author to self.author
    # methods
    def info(self):
        # returns information about book when it is called
        return f'{self.title} by {self.author}'

# Now we can create book objects from base book class
book1 = Book('The Alchemist','Paulo Coelho')
book1.title # 'The Alchemist'
book1.info() # 'The Alchemist by Paulo Coelho'
# We can create many objects from same class
book2 = Book('Harry Potter 1','J.K. Rowling')
book3 = Book('Odyssey','Homer')
```

You can also edit object properties like : book2.title = “Harry Potter 2”

Python Package Manager - PIP

What is PIP ?

https://pypi.org/



```
pip install django
pip uninstall django
```

Useful standard library

- RE
- Random
- Datetime
- OS
- Tkinter
- Math
- CSV

Useful pip packages

- Pipenv
- BeautifulSoup
- Django
- Flask
- Pygame
- Numpy
- Tensorflow