**Python:**

**Pythion is a dynamic typed language and executes at runtime. While Java and c are static typed language meaning java and c language are compiled before runtime. Most of time python uses only one CPU while java uses multi-CPU to process the application.**

# **Introduction**

## *About*

# **Variables**

## *Local variable*

## *Global variable*

## *Class variable*

# **Data Types**

## *String*

## *Float*

## *Int*

## *Booleans*

## *Lists*

## *Dictionary*

## *Set*

## *Tuple*

# **Operators**

## *Mathematical operations*

## *Equals*

## *None*

## *Is vs ==*

# **Control Flow**

## *If else*

# **Loops**

## *For loop*

## *While loop*

## *For each*

# **Try Catch**

# **Comprehensions**

# **Functions**

# **Decorators**

# **OOPs**

# **Collections**

# **Iterator**

# **Built In**

# **Args kwargs**

# **Generator**

# **Date Time**

# **Regular Expression**

# **Exceptions**

# **Miscellaneous**

## *Higher order Function*

## *Positional argument*

## *Binary search*

## *Linear search*

## *Function argument unpack*

# Variable:

## *Class variable:*

Class variable contains data that is shared among all the instances of class

# Python Data Type:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Integers | int | Whole numbers such as 3, 300 |
| Floating point | float | Numbers with decimal point 2.3, 4.6 |
| Strings | str | Ordered sequence of characters: “hello”, “name” |
| Lists | list | Ordered sequence of objects: [10, “hello”, 20.4] |
| Dictionaries | dict | Unordered key: value pairs: {“myKey”: “value”, “name”: “Frank”} |
| Tuples | tup | Ordered immutable sequence of objects: (10, “hello”,20.4) |
| Sets | set | Unordered collection of unique objects: {“a”, “b”} |
| Booleans | bool | Logical value indicating true or false |

## *String*

### Strings are immutable

name = **'Shahjalal'**sentence = **"my name is Shahjalal"**print (name[0])  
  
**for** char **in** name:  
 print(char)  
  
**# get all character except last one**  
print(name[:-1])  
print(name[-1])  
  
**#reversing the whole string**  
print(name[::-1])  
  
# cannot replace a character once the string is created  
  
name[0] = **"X"**

**# length of character**  
print(len(name))  
  
**# multiplying the letter**  
letter = **'z'**print (letter \* 10)

# **uppercase**  
print(name.upper())  
  
# **lowercase**  
print(name.lower())  
  
# **Split the string**  
print(name.split(**'a'**))

Print(name.

### **Advance String method**

name = 'shahjalal'  
  
*#*Capitalize -- means capitalize the the first letterprint(name.capitalize())  
  
# counting the particular letter from a stringprint(name.count('a')) *# a came three times in 'shahjalal' String*# Check if the string is lower caseprint(name.islower())  
  
# Check if the string is upper caseprint(name.isupper())  
  
# Check if the string has any spaceprint(name.isspace())  
  
# Check if the string is alpha numericprint(name.isalnum())  
  
# Check if the string is only alphabiticprint(name.isalpha())  
  
# Check the last character is equal to somethingprint(name.endswith('l'))  
print(name.endswith('p'))  
  
# partitionprint(name.partition('a')) *# it will partition the string with first occurence with a*

# rfind

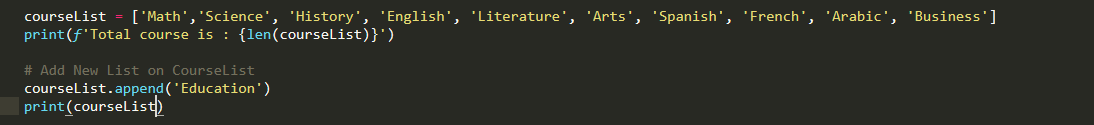
#Find the last position of a substring “Emma” in a given string  
Given:  
'''  
  
str1 = "Emma is a data scientist who knows Python. Emma works at google. Emma"  
index = str1.rfind("Emma")

## *List:*

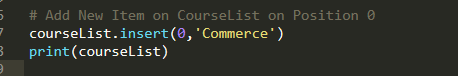
A list is a collection which is ordered and changeable. In Python lists are written with square brackets.



*To Add New Item in a list: Use append. By Default item will be added in end of the list*



*To Add New Item on Certain position in a list: Use Insert.*



### Append VS Extend

[**Append**](https://www.geeksforgeeks.org/list-methods-python/)**:** Adds its argument as a single element to the end of a list. The length of the list increases by one. Append is used to add single item in a list. If you append another list onto a list, the parameter list will be a single object at the end of the list.

*my\_list = ['geeks', 'for', 'geeks']*

*another\_list = [6, 0, 4, 1]*

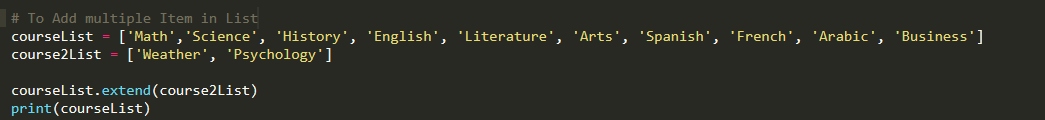
*my\_list.append(another\_list)*

*print my\_list*

*['geeks', 'for', 'geeks', [6, 0, 4, 1]]*

**Extend** is used to add multiple item in a list. if you extend a list with a string, you’ll append each character as you iterate over the string.

Extend: To Add Multiple item in a list Use Extend.



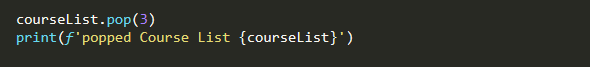
### Remove:

The remove () method removes the specified item:



### Pop:

The pop () method removes the specified index, (or the last item if index is not specified):



### Del:

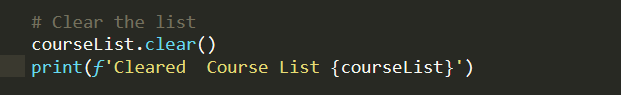
The del keyword removes the specified index:



The del keyword can also delete the list completely:



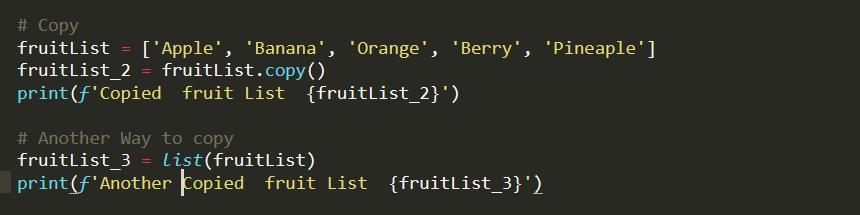
### Clear:

The clear () method empties the list:

### Copy:

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

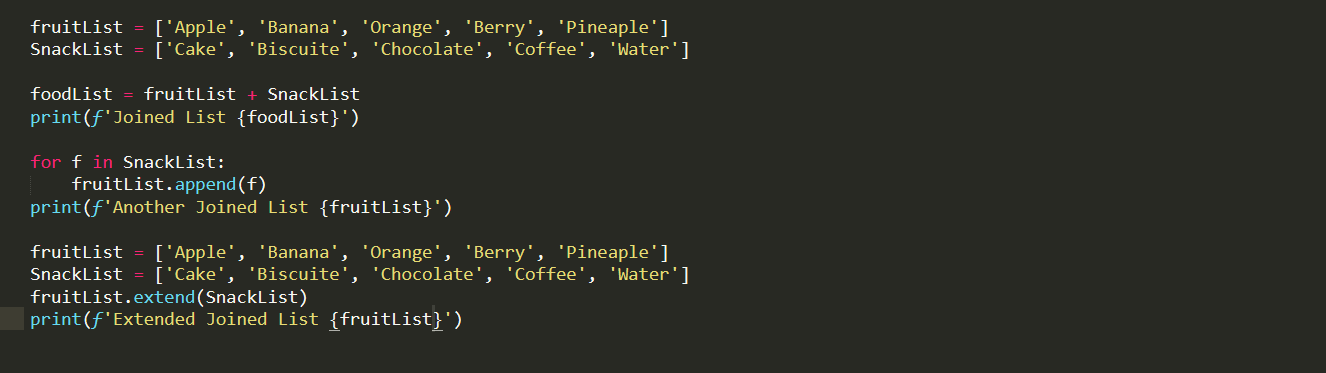
*There are ways to make a copy, one way is to use the built-in List method copy ().*

` 

### Joining:

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator, extend and append



### Sort

List.sort() – orginal list will be sorted

Newlist = sorted(oldlist) – old lists will be same and new list will be sorted

## Tuples:

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

*tup1 = ('physics', 'chemistry', 1997, 2000);*

*tup2 = (1, 2, 3, 4, 5);*

*tup3 = "a", "b", "c", "d";*

The empty tuple is written as two parentheses containing nothing −

tup1 = ();

To write a tuple containing a single value you have to include a comma, even though there is only one value −

tup1 = (50,);

Accessing Values in Tuples:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

tup1 = ('physics', 'chemistry', 1997, 2000);

tup2 = (1, 2, 3, 4, 5, 6, 7);

print "tup1[0]: ", tup1[0]

print "tup2[1:5]: ", tup2[1:5]

*A tuple value can be changed via changing variable value, this is a sneaky way to do it.*

tuple\_1 = ('Math', 'Art', 'Music', 'Physics')

print(tuple\_1)

print(type(tuple\_1))

***# Accessing the Tuple***

print(tuple\_1[1])

***# Print the last item of tuple***

print(tuple\_1[-1])

***# change the tuple values***

list\_tuple = list(tuple\_1)

print(list\_tuple)

list\_tuple[1] = 'Education'

print(list\_tuple)

tuple\_1 = tuple(list\_tuple)

print(tuple\_1)

***# loop thru a tuple***

for t in tuple\_1:

print(t)

***# Check if tuple item is exist***

if 'Math' in tuple\_1:

print('Tuple Exist')

***# Delete Tuple***

del tuple\_1

***# Join two tuple***

this\_tuple = ('Math', 'English')

that\_tuple = ('Science', 'Art')

join\_tuple = this\_tuple + that\_tuple

print(join\_tuple)

***# Count: Return the number of times of an item appears in the tuple***

num\_tuple = (1,3,9,8,8,7,5,4,3,2,5,6,8)

print(num\_tuple.count(8))

## List vs Tuples:

* *List consists of homogeneous data – either string or integer or any other type [ ‘a’, ’b’, ‘c’] or [1, 2, 3,]*
* *Tuples consists of heterogeneous data – (‘a’, ’b’, ‘c’, 1, 2, 3,)*
* *List item can be updated*
* *Tuples cannot be updated, they are fixed*

## *Dictionary:*

*Dictionary is unordered collections of data and guaranteed for no duplicate data. Dictionary contain key value pair. Dictionary value cannot be accessed by index, it is accessed by key.*

*fruit = {"orange": "a citrus fruit",  
 "apple": "good for making cider",  
 "lemon": "Sour, a yellow citrus fruit",  
 "grape": "a sweet fruit",  
 "lime": "a sour, green citrus fruit"}*

*A dictionary can hold different type of object.*

*car={"color": "Red", "Cylinder": 4}*

*Dictionary cannot be added or appended. To add a new key, new key must be assigned.*

### **Add new key:**

*fruit["pear"] = "an odd shape apple"*

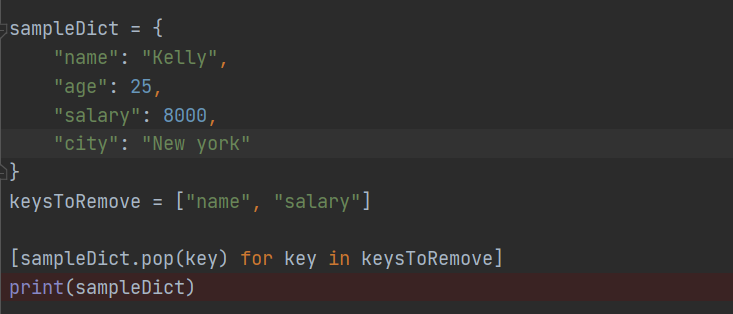
*keys are in dictionary is Unique. If a new value is assigned in an existing key, new value will replace old key rather than creating a new entry.*

*fruit["lime"] = "a great stuff"  
print(fruit)*

### Remove Key, del keyword

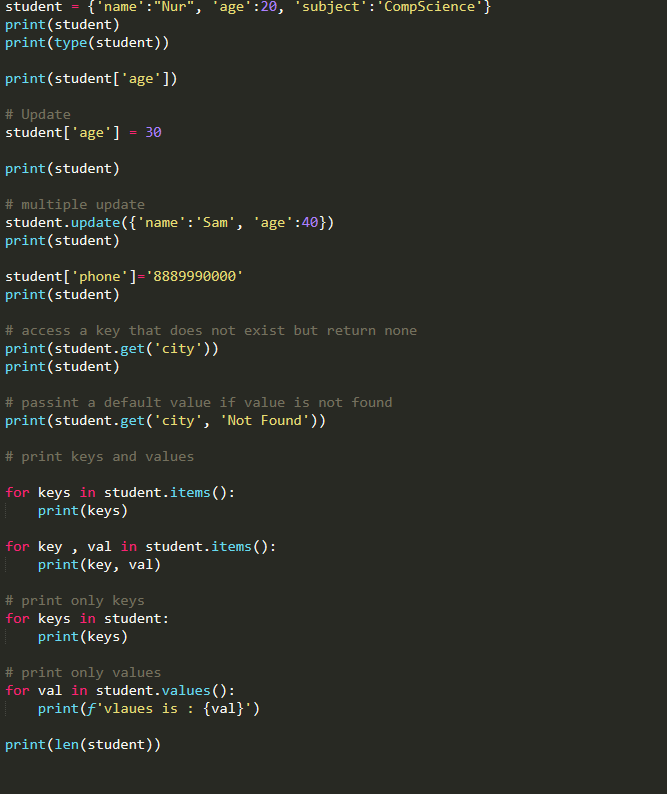
*del fruit["lemon"]  
print(fruit)*

### Delete set of keys

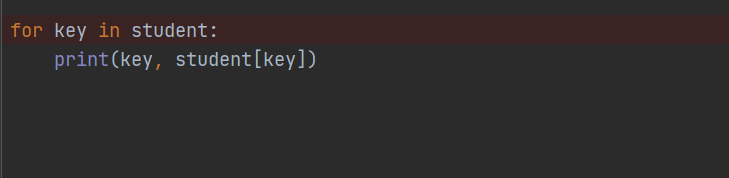


*To delete all entry but keep an empty dictionary*

*fruit.clear()  
print(fruit)*

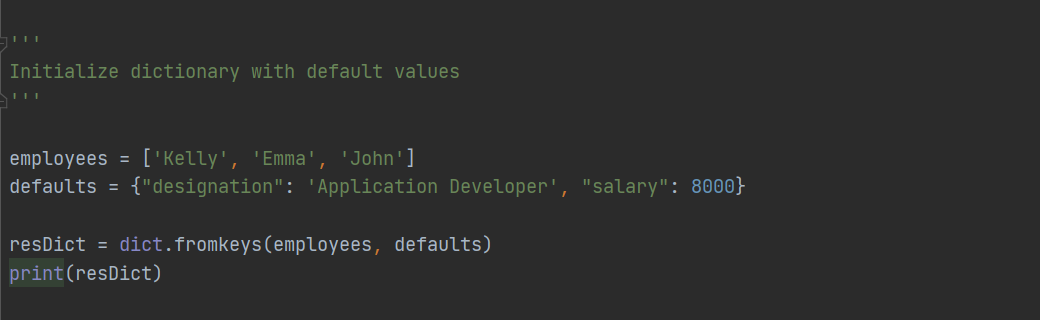


### Get key , val



### Fromkey:

fromkeys() method is used to **create a new dictionary from a given sequence of keys and a defined value by the user**.



## *Set*

### **Sets in Python**

*A set contains an*

* *unordered collection of*
* *unique and*
* *immutable objects.*

### **Empty Set**

*empty\_Set = set ()*

### **Add New Element**

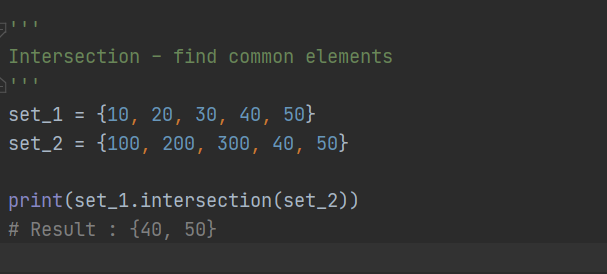
*OceanFish.add("Shark")*

*print(OceanFish)*

### **Intersection**

**Matching between two sets or common set**

*print(set\_1.intersection(set\_2))*

**

### **Intersection\_update**

### **Union**

**Unique matching and un matching**

*print(set\_1.union(set\_2))*

### **Difference**

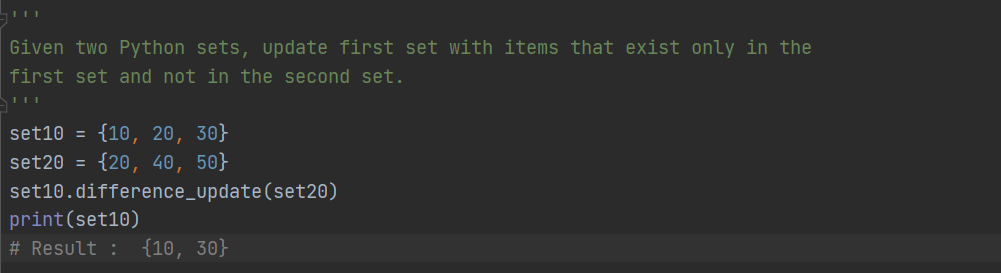
**Difference from comparing set**

*print(set\_1.difference(set\_2)) # difference of set\_1 from set\_2*

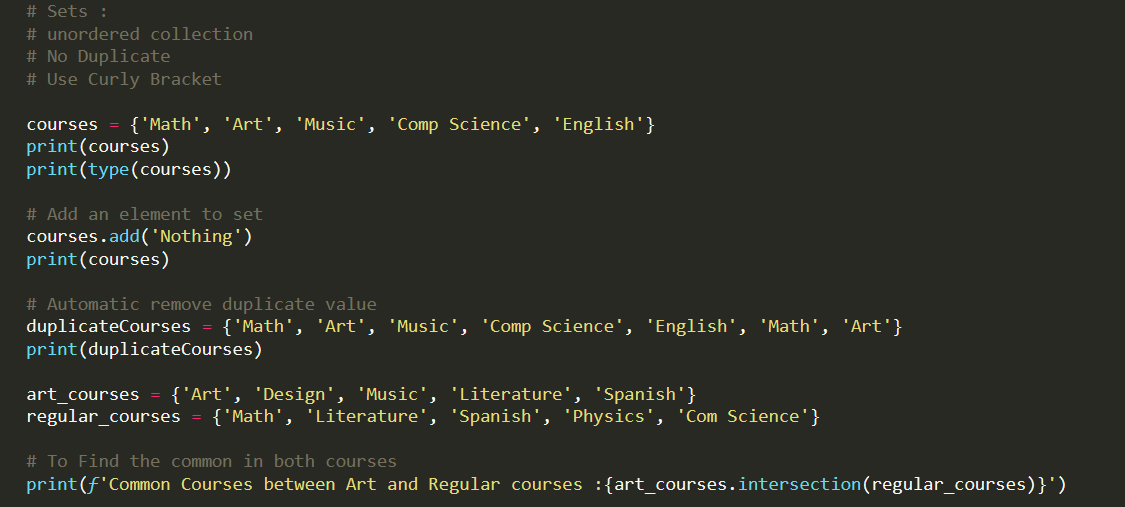
*print(set\_2.difference(set\_1)) # difference of set\_2 from set\_1*

### **Difference\_update**

Remove common item from target set.



### **symmetric\_difference**





## Shelve*:*

## Repr

**repr**() : evaluatable string representation of an object (can "eval()" it, meaning it is a string representation that evaluates to a **Python** object) In other words: >>> x = 'foo' >>> **repr**(x) "'foo'

## Python Exception Handling:

*Python provides two important features to handle any unexpected error.*

1. *Exception Handling*
2. *Assertions*

*Exception Name:*

# Exception # StopIteration # SystemExit #StandardError # ArithmeticError

# EOFError # FloatinPointError # FloatinPointError # ZeroDivisionError #StandardError #ArithmeticError # Exception # StopIteration # SystemExit #StandardError # ArithmeticError # AssertionError # AttributeError # *EnvironmentError*

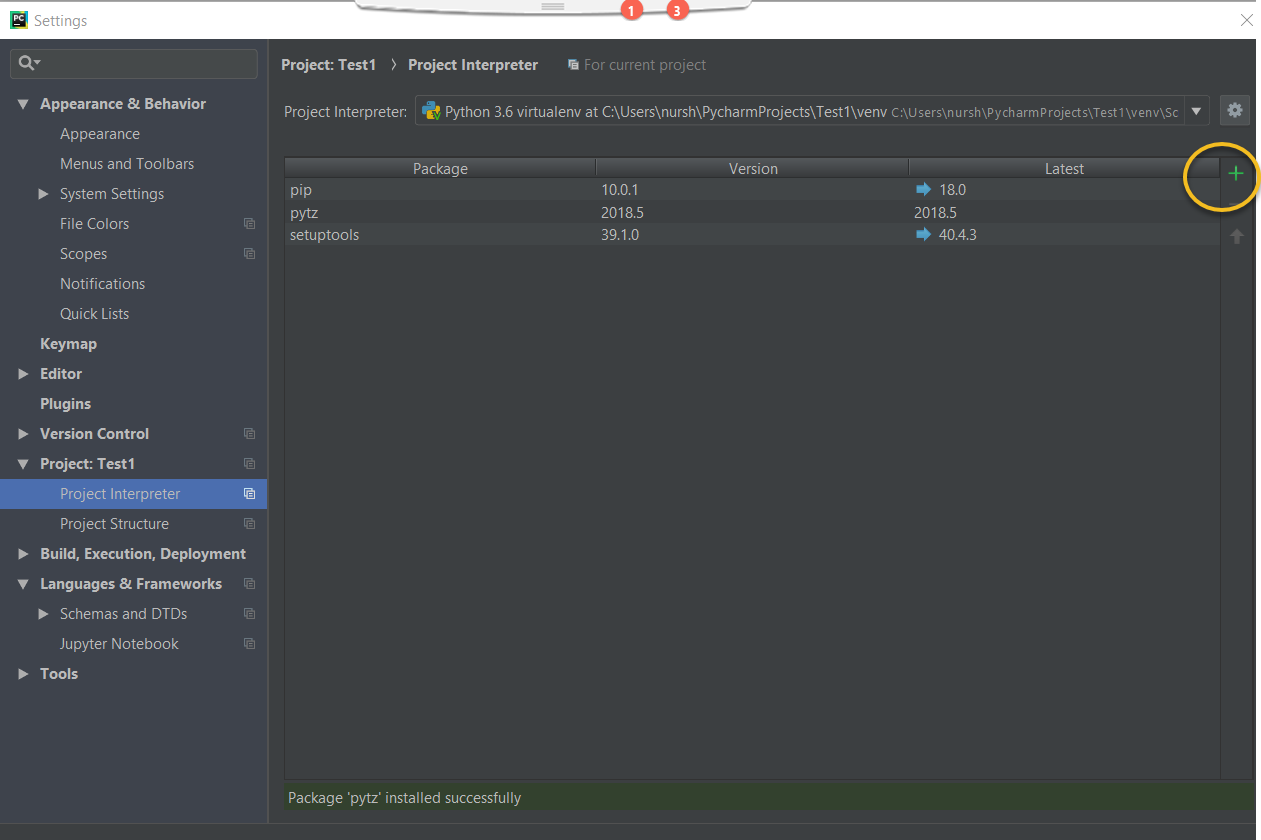
### Install PYTZ module in PYCharm

*Pip install pytz*

*If import pytz gives an error in pycharm:*

*Go to File >> Settings >> Project >> Project Interpreter >>*

*Click on + sign and choose pytz module to install*



## Static Method

@staticmethod means: when this method is called, we don't pass an instance of the class to it (as we normally do with methods). This means you can put a function inside a class but you can't access the instance of that class (this is useful when your method does not use the instance).

*@staticmethod  
A****staticmethod****is a method that knows nothing about the class or instance it was called on. It just gets the arguments that were passed, no implicit first argument. It is basically useless in Python -- you can just use a module function instead of a staticmethod.*

@staticmethod function is nothing more than a function defined inside a class. It is callable without instantiating the class first. Its definition is immutable via inheritance.

## Magic Method / Dunder Method

There are some special method in python surrounded by double underscore(\_\_). Most two popular are:

\_\_str\_\_

\_\_repr\_\_

## \*args and \*kargs

\*args and \*\*kwargs are mostly used in function definitions. \*args and \*\*kwargs allow you to pass a variable number of arguments to a function. What does variable mean here is that you do not know before hand that how many arguments can be passed to your function by the user so in this case you use these two keywords. \*args is used to send a non-keyworded variable length argument list to the function.

# Regular method with \*args  
def car(\*args):  
 print(args)  
  
  
# passing as many arguments with the help of \*args syntax  
car("blue", "Four Cylinder", "AWD")  
# result : ('blue', 'Four Cylinder', 'AWD')

\*\*kwargs allows you to pass keyworded variable length of arguments to a function. You should use \*\*kwargs if you want to handle named arguments in a function. Here is an example to get you going with it:

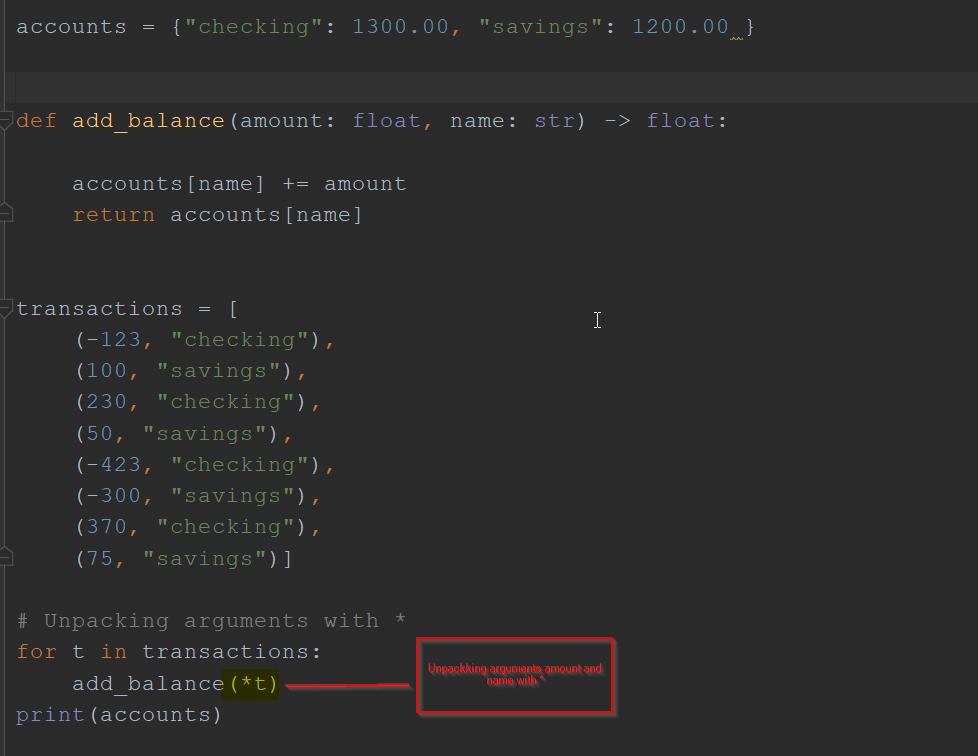
def newcar(\*args, \*\*kwargs):  
 print(args,kwargs)  
  
  
newcar("Black", "Two Door", "6 cylinder" , horsepower=360, model="Tesla")  
# result : ('Black', 'Two Door', '6 cylinder') {'horsepower': 360, 'model': 'Tesla'}

## Argument Unpacking

There are 4 cases for using the asterisk in Python.

* For multiplication and power operations.
* For repeatedly extending the list-type containers.
* For using the variadic arguments. (so-called “packing”)
* For unpacking the containers.

We use two operators \* (for tuples) and \*\* (for dictionaries).



of list comprehension  
squires = [i\*i for i in range(10)]  
print(squires)

Values = [Expression For Value in Collection]

***The Syntax:***

*[ Expression \_ for \_ In ]*

## Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression. The expression is evaluated and returned. Lambda functions can be used wherever function objects are required.

Do not use def or return keyword. They are implicit.

lambda arguments: expression

# same thing can be achieved using lambda function  
val = lambda x: 2 \* x  
add = lambda x, y: x+y  
max = lambda x, y: x if x> y else y  
  
  
print(val(3))  
print(add(3, 7))  
print(max(1000, 90000))

Simply reduce function takes a list and return an item based the action of the function.

## The Difference between Yield and Return

The keyword return returns a value from a function, at which time the function then loses its local state. Thus, the next time we call that function, it starts over from its first statement.

On the other hand, yield maintains the state between function calls, and resumes from where it left off when we call the next() method again. So if yield is called in the generator, then the next time the same generator is called we'll pick right back up after the last yield statement.

## Using return in a Generator

A generator ***can*** use a return statement, but only without a return value, that is in the form:

return

When the generator finds the return statement, it proceeds as in any other function return.

## Python Generator:

A **Python generator** is a function that produces a sequence of results. It works by maintaining its local state, so that the function can resume again exactly where it left off when called subsequent times. Thus, you can think of a **generator** as something like a powerful iterator.

**Generator** functions allow you to declare a function that behaves like an iterator.

**Generator** is a special function that remembers its state in between execution.

type of object called a **generator**, which allows us to generate arbitrarily-many items in a series, without having to store them all in memory at once.

## Csv reading and Writing

import csv  
  
with open('names.csv', 'r') as csv\_file:  
 csv\_reader = csv.reader(csv\_file)  
  
 # for line in csv\_reader:  
 # print(line)  
  
 for line1 in csv\_reader:  
 print(line1[0])

import csv  
  
with open("names.csv", "r") as csv\_file:  
  
 csv\_reader = csv.DictReader(csv\_file)  
  
 # for line in csv\_reader:  
 # print(line)  
  
 with open("new\_dict\_writer.csv", "w") as new\_file:  
 fieldnames = ["first\_name", "last\_name"]  
 csv\_writer = csv.DictWriter(new\_file, fieldnames=fieldnames, delimiter='\t')  
  
 csv\_writer.writeheader()  
  
 for line1 in csv\_reader:  
 del line1["email"]  
 csv\_writer.writerow(line1)

## Remove Duplicate Word

# set will remove duplicate  
  
given\_word = 'hello world and practice makes perfect and hello world again'  
sorted\_word = []  
for i in given\_word.split(" "):  
 sorted\_word.append(i)  
  
print(sorted(set(sorted\_word)))

Problems:

**Find most common words from a list:**

Names = ['abul', 'babul', 'kabul', 'chulbul', 'abul', 'Babul', 'Nupur', 'nupur', 'kabul', 'abul', 'babul', 'kabul', 'chulbul', 'abul', 'Babul', 'Nupur', 'nupur', 'kabul', 'Kabita']

print(Counter(names).most\_common(1))

# most\_common(1), 1 specifies the number of items to print

for name in Counter(names).most\_common(1):  
 print(name[0])

**Find first duplicate word from a list:**

Names = ['abul', 'babul', 'kabul', 'chulbul', 'abul', 'Babul', 'Nupur', 'nupur', 'kabul', 'abul', 'babul', 'kabul', 'chulbul', 'abul', 'Babul', 'Nupur', 'nupur', 'kabul', 'Kabita']

dup\_word = set()  
  
for item in dup\_names:  
 if item in dup\_word:  
 print(f'First Duplicate word is { item}')  
 break  
 dup\_word.add(item)

**Find all duplicate words from a list:**

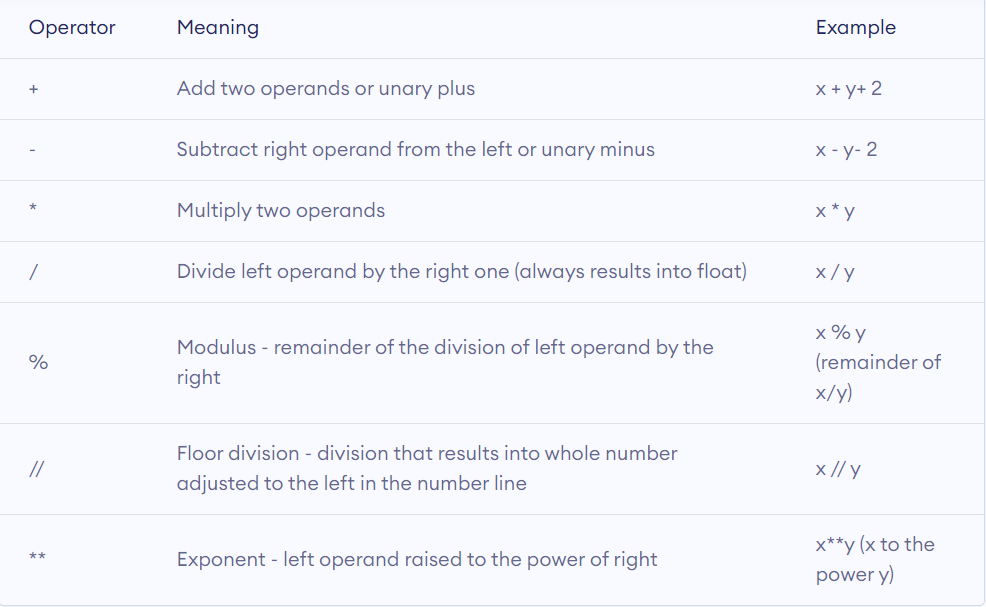
Names = ['abul', 'babul', 'kabul', 'chulbul', 'abul', 'Babul', 'Nupur', 'nupur', 'kabul', 'abul', 'babul', 'kabul', 'chulbul', 'abul', 'Babul', 'Nupur', 'nupur', 'kabul', 'Kabita']

**second solutions:**

unique\_list = []  
for x in dup\_names:  
 if dup\_names.count(x) > 1:  
 unique\_list.append(x)  
  
print(set(unique\_list))

# Python Operator

## *Arithmetic Operator*

**

# Control Flow

## *IF ELSE*

# Loops

# Comprehensions

## *List Comprehension*

 List comprehension is an elegant way to define and create lists based on existing lists.

# print squire number of given number  
# with regular way with for loop  
  
emptylist = []  
for i in range(10):  
 emptylist.append(i\*i)  
print(emptylist)  
  
  
print("==" \* 30)  
# same thing can be achieved in one line with the help

## *Dictionary Comprehension*

## *Tuple Comprehension*

## *Set Comprehension*

# Functions

# Decorators

# OOPs

## *Inheritance*

## *Self*

## *Class method*

## *Class Variable*

## *Instance variable*

## *Static method*

## *Class initializations*

## *Magic Method/Dunder Method*

### \_\_init\_\_

### \_\_repr\_\_

### Getter

### Setter

## Python Class – Object Oriented Programming

Object oriented programing is used to help conceptualized the interactions between objects.

A class allows us to logically group data and function. A class is a blueprint of creating instances.

**What is object:**

An object is a state that holds data and action.

## Self-Keyword

Why we need self-keyword, self is passed automatically in the instance of a class when we create an instance. It is the part of the class.

## What is Self in Python

**self** represents the instance of the class. By using the "**self**" **keyword** we can access the attributes and methods of the class in **python**

## Instance Variable:

Instance variable contains data that is unique to each instance of class.

*class Employee:  
 # init\_\_ is the default constructor  
 def \_\_init\_\_(self, first\_name, last\_name, pay):  
 self.first\_name = first\_name  
 self.last\_name = last\_name  
 self.pay = pay  
 self.email = first\_name + '.' + last\_name + '@email.com'  
  
 def fullname(self):  
 return self.first\_name + ' ' + self.last\_name  
  
  
# instance variable are passing  
emp\_1 = Employee('Nur', 'Shahjalal', '90000')  
emp\_2 = Employee('Test', 'User', '80000')  
  
print(emp\_1.email)  
print(emp\_2.email)  
  
print(emp\_1.fullname())  
print(emp\_2.fullname())  
  
# calling directly from class  
fl\_name = Employee.fullname(emp\_1)  
print(fl\_name)*

## Class Method:

*A class method receives the class as implicit first argument, just like an instance method receives the instance.*

@classmethod*means: when this method is called, we pass the class as the first argument instead of the instance of that class (as we normally do with methods). This means you can use the class and its properties inside that method rather than a particular instance.*

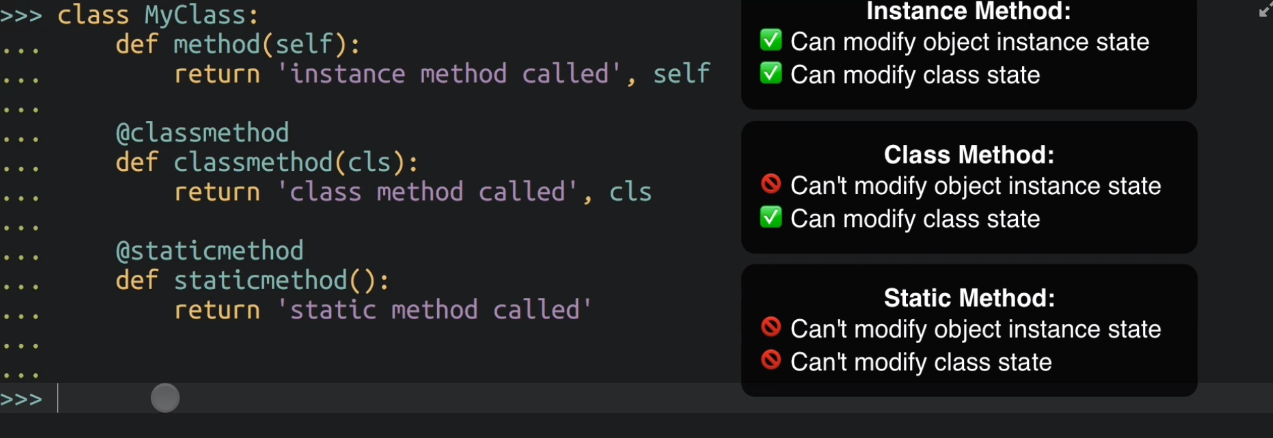
@classmethod

*A****classmethod****, on the other hand, is a method that gets passed the class it was called on, or the class of the instance it was called on, as first argument. This is useful when you want the method to be a factory for the class: since it gets the actual class it was called on as first argument, you can always instantiate the right class, even when subclasses are involved.*

***When to use what?***

*We generally use class method to create factory methods. Factory methods return class object ( similar to a constructor ) for different use cases.*

*We generally use static methods to create utility functions.*



# Collections

## Counter

*A*[***Counter***](https://docs.python.org/2/library/collections.html#collections.Counter)*is a****[dict](https://docs.python.org/2/library/stdtypes.html" \l "dict" \o "dict)****subclass for counting hashable objects. It is an unordered collection where elements are stored as dictionary keys and their counts are stored as dictionary values. Counts are allowed to be any integer value including zero or negative counts. The*[***Counter***](https://docs.python.org/2/library/collections.html#collections.Counter)*class is similar to bags or multisets in other languages.*

*Elements are counted from an*iterable*or initialized from another*mapping*(or counter):*

***>>>*** *c = Counter()* # a new, empty counter

***>>>*** *c = Counter('gallahad')* # a new counter from an iterable

***>>>*** *c = Counter({'red': 4, 'blue': 2})* # a new counter from a mapping

***>>>*** *c = Counter(cats=4, dogs=8)* # a new counter from keyword args

*Counter objects have a dictionary interface except that they return a zero count for missing items instead of raising a*[***KeyError***](https://docs.python.org/2/library/exceptions.html#exceptions.KeyError)*:*

***>>>*** *c = Counter(['eggs', 'ham'])*

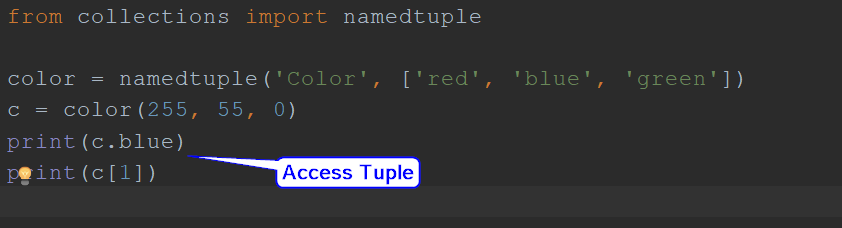
***>>>*** *c['bacon']* # count of a missing element is zero

*0*

## NamedTuple

Named Tuples in Python are **Immutable** High-performance container datatypes. Sometimes we need something between a tuple and a dictionary. Tuples have less typing to do but it is easy to forget what each value stands for. In contrast, Dictionaries are very wordy, while we can assign the keys to what the value represents.

NamedTuple returns a class (that's a child of the built-in class tuple). The first argument you pass to NamedTuple becomes the name of the class, while the list of strings becomes the attributes (data fields). You can then call the constructor (the line of code before the print() method in the video) to make objects.



## Defaultdict

<https://realpython.com/python-defaultdict/>

A common problem that you can face when working with Python [dictionaries](https://realpython.com/python-dicts/) is to try to access or modify keys that don’t exist in the dictionary. This will raise a [KeyError](https://realpython.com/python-keyerror/) and break up your code execution. To handle these kinds of situations, the [standard library](https://docs.python.org/3/library/index.html) provides the Python **defaultdict** type, a dictionary-like class that’s available for you in [collections](https://docs.python.org/3/library/collections.html#module-collections).

The Python [defaultdict](https://docs.python.org/3/library/collections.html#collections.defaultdict) type behaves almost exactly like a regular Python dictionary, but if you try to access or modify a missing key, then defaultdict will automatically create the key and generate a default value for it. This makes defaultdict a valuable option for handling missing keys in dictionaries.

*Python defaultdict type does two things:*

1. *It overrides*[*.\_\_missing\_\_()*](https://docs.python.org/3/library/collections.html#collections.defaultdict.__missing__)*.*
2. *It adds .default\_factory, a writable instance variable that needs to be provided at the time of instantiation.*

Sometimes, you’ll use a mutable built-in collection (a list, dict, or [set](https://realpython.com/python-sets/)) as values in your Python dictionaries. In these cases, you’ll need to **initialize the keys** before first use, or you’ll get a KeyError. You can either do this process manually or automate it using a Python defaultdict.

*Python defaultdict type for solving some common programming problems:*

* ***Grouping****the items in a collection*
* ***Counting****the items in a collection*
* ***Accumulating****the values in a collection*

## Deque

Doubly ended list.

## OrderDict

An order dict is a dictionary subclass remembers the order in which the contents are added.

# Iterator

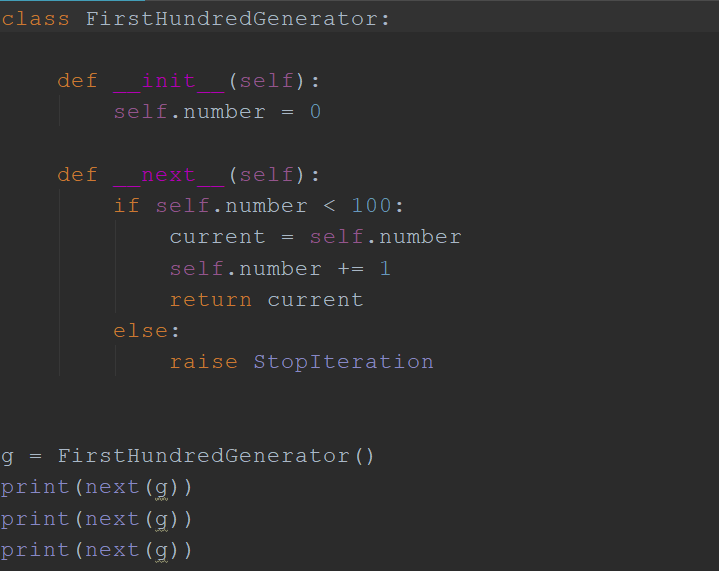
## Iterator

Iterator is used to get the next value

An iterator is an object that contains a countable number of values.

Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the method \_\_next\_\_().

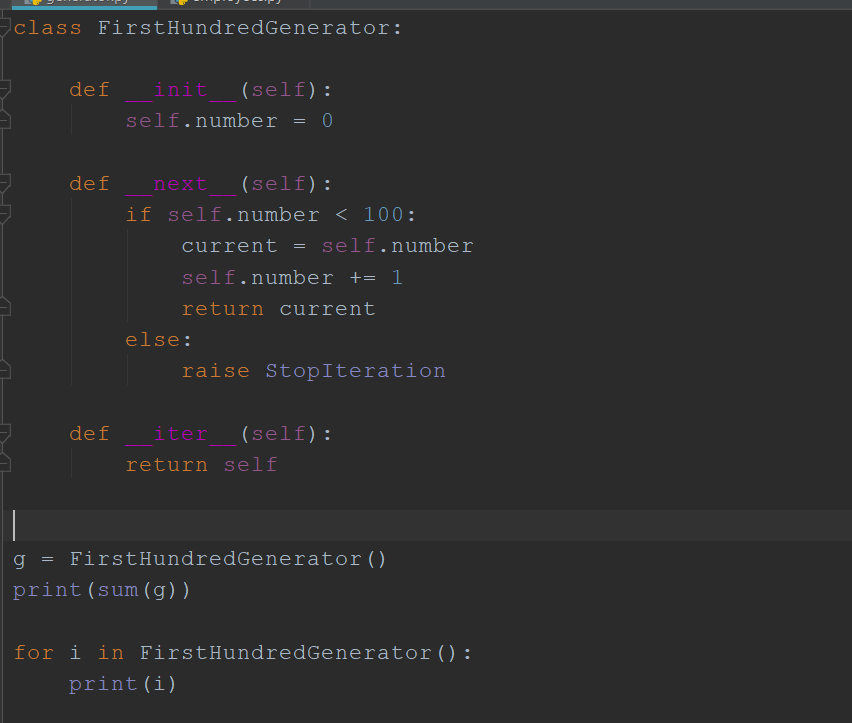
If a class has a \_\_next\_\_() method , that class is iterator not Iterable.



## Iterable

Iterable is used to go over all the

Any object that consist of \_\_iter\_\_ method is iterable.



# Python Built in Functions

* Zip
* Map
* Filter
* Lambda
* Enumerate
* Random
* any
* Time
* Date

## *map ()*

The map () function applies a given function to each item of an iterable (list, tuple etc.) and returns a list of the results.

Map takes an iterator and return an iterator

Map (function, iterator)

Function as first argument, iterator as second argument

# returns a list of the results.  
  
def calculateS\_suire(n):  
 return n\*n  
  
numbers = (1, 2, 3, 4)  
result = map(calculateS\_suire, numbers)  
print(list(result))  
  
def length\_of\_letter(n):  
 return len(n)  
  
letters = ("apple", "Banana", "Avocado", "New York")  
ln = map(length\_of\_letter, letters)  
print(list(ln))

## *Filter*

The filter () function returns an iterator where the items are filtered through a function to test if the item is accepted or not.

Filter takes two arguments; first one is a function and second one is data[iterator]. And it returns an iterator object. To print use list, set or dictionary

num = [1, 2, 3, 4, 5, 6, 7, 8, 90, 67, 43, 44, 76, 32, 78, 21, 89, 0]  
  
even\_num = filter(lambda x: x%2==0, num)  
print(list(even\_num))  
  
words = ["Apple", "cat", "Bowling", "New", "adam"]  
upper\_word = filter(lambda x: x[0]== "A", words)  
print(list(upper\_word))

## *Reduce*

The reduce (fun, seq) function is used to apply a particular function passed in its argument to all of the list elements mentioned in the sequence passed along.

Reduce returns an item not a list.

Apply function of two arguments cumulatively to the items of sequence, from left to right, so as to reduce the sequence to a single value. For example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]) calculates ((((1+2)+3)+4)+5). The left argument, x, is the accumulated value and the right argument, y, is the update value from the sequence. If the optional initializer is present, it is placed before the items of the sequence in the calculation, and serves as a default when the sequence is empty. If initializer is not given and sequence contains only one item, the first item is returned.

## *Lambda*

We use lambda functions when we require a nameless function for a short period of time.

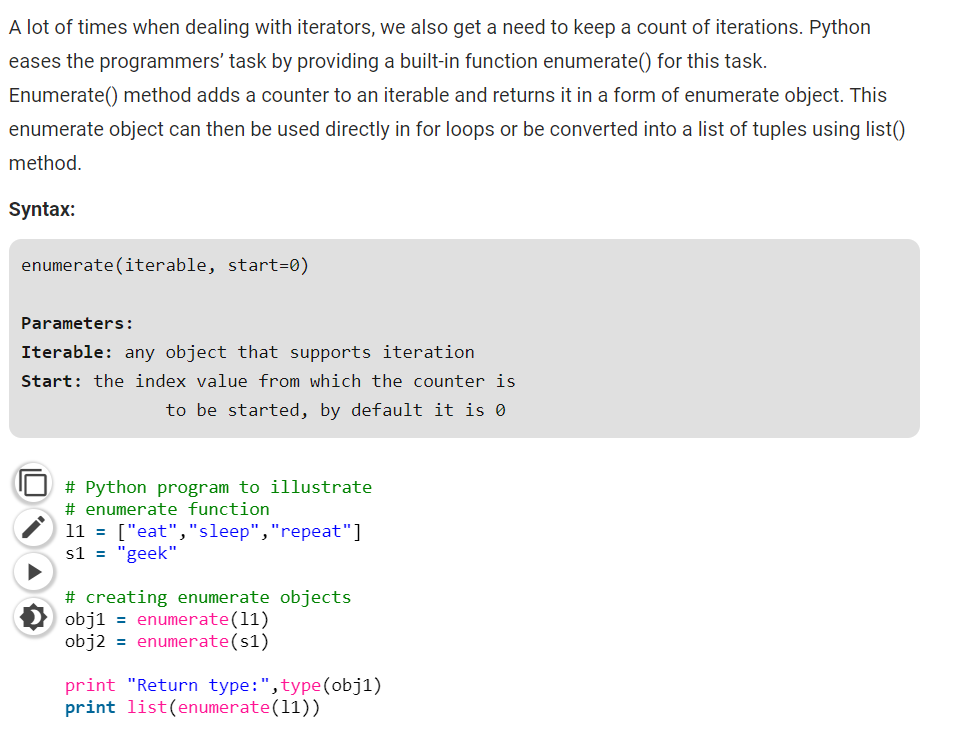
**Syntax of Lambda Function in python:**

lambda arguments: expression

In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as [arguments](https://www.programiz.com/python-programming/function-argument)). Lambda functions are used along with built-in functions like filter(), map() etc.

|  |
| --- |
| **Ex: 1**  Even\_or\_Odd = lambda x : x % 2 == 0  Print(Even\_or\_Odd(4)) # returns true  **Ex:2**  Multiply\_num = lambda x: x \* 2  Print(Multiply\_num(5)) # returns 10 |

## *Enumerate*



## *Any*

## Any and All

Any : Returns true if any of the items is True. It returns False if empty or all are false. Any can be thought of as a sequence of OR operations on the provided iterables.  
It short circuit the execution i.e. stop the execution as soon as the result is known.

Syntax : any(list of Iterable)

## *All*

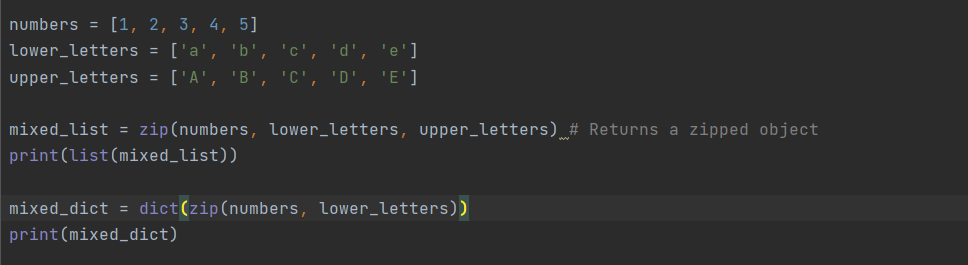
## *Zip*

Python’s zip function is defined as zip(\*Iterable). The function takes in Iterable as arguments and returns an iterator. This iterator generates a series of tuples containing elements from each Iterable. zip can accept any type of Iterable, such as files, lists, tuples, dictionaries, sets, and so on.

### **Syntax:**

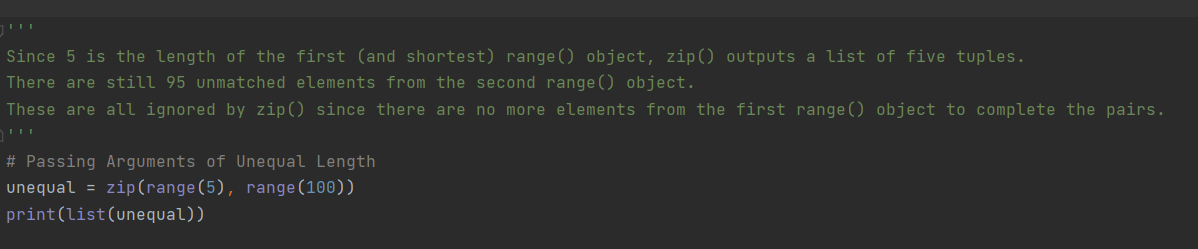
|  |
| --- |
| **zip(\*Iterable)**  # takes n number of arguments |

### **Zip Example**

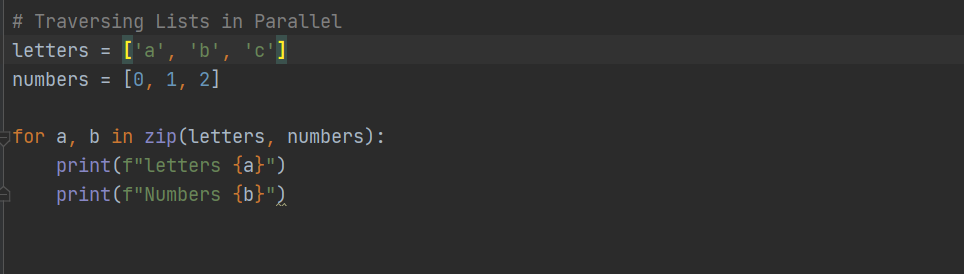


### **Unequal length of arguments:**

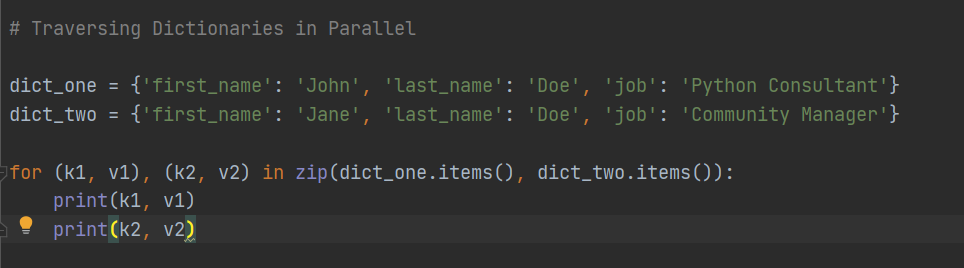
Zip will unpack the shortest length of arguments and rest will be ignored.



### **Traversing lists in parallel:**

****

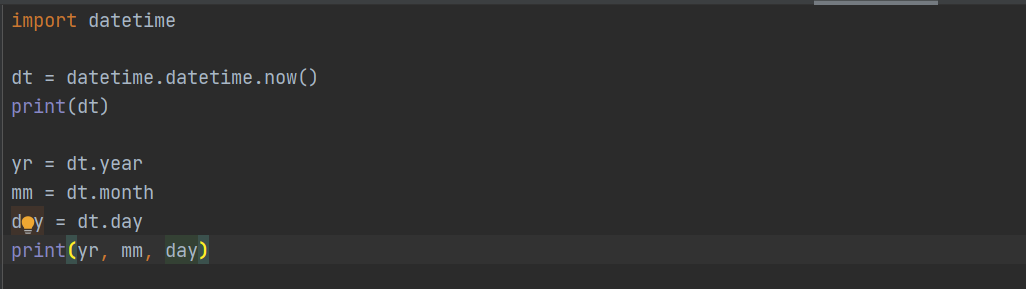
### **Traversing Dictionaries in Parallel**



|  |  |
| --- | --- |
| ***Function*** | ***Description*** |
| [*abs()*](https://www.w3schools.com/python/ref_func_abs.asp) | *Returns the absolute value of a number* |
| [*all()*](https://www.w3schools.com/python/ref_func_all.asp) | *Returns True if all items in an iterable object are true* |
| [*any()*](https://www.w3schools.com/python/ref_func_any.asp) | *Returns True if any item in an iterable object is true* |
| [*ascii()*](https://www.w3schools.com/python/ref_func_ascii.asp) | *Returns a readable version of an object. Replaces none-ascii characters with escape character* |
| [*bin()*](https://www.w3schools.com/python/ref_func_bin.asp) | *Returns the binary version of a number* |
| [*bool()*](https://www.w3schools.com/python/ref_func_bool.asp) | *Returns the boolean value of the specified object* |
| [*bytearray()*](https://www.w3schools.com/python/ref_func_bytearray.asp) | *Returns an array of bytes* |
| [*bytes()*](https://www.w3schools.com/python/ref_func_bytes.asp) | *Returns a bytes object* |
| [*callable()*](https://www.w3schools.com/python/ref_func_callable.asp) | *Returns True if the specified object is callable, otherwise False* |
| [*chr()*](https://www.w3schools.com/python/ref_func_chr.asp) | *Returns a character from the specified Unicode code.* |
| *classmethod()* | *Converts a method into a class method* |
| [*compile()*](https://www.w3schools.com/python/ref_func_compile.asp) | *Returns the specified source as an object, ready to be executed* |
| [*complex()*](https://www.w3schools.com/python/ref_func_complex.asp) | *Returns a complex number* |
| [*delattr()*](https://www.w3schools.com/python/ref_func_delattr.asp) | *Deletes the specified attribute (property or method) from the specified object* |
| [*dict()*](https://www.w3schools.com/python/ref_func_dict.asp) | *Returns a dictionary (Array)* |
| [*dir()*](https://www.w3schools.com/python/ref_func_dir.asp) | *Returns a list of the specified object's properties and methods* |
| [*divmod()*](https://www.w3schools.com/python/ref_func_divmod.asp) | *Returns the quotient and the remainder when argument1 is divided by argument2* |
| [*enumerate()*](https://www.w3schools.com/python/ref_func_enumerate.asp) | *Takes a collection (e.g. a tuple) and returns it as an enumerate object* |
| [*eval()*](https://www.w3schools.com/python/ref_func_eval.asp) | *Evaluates and executes an expression* |
| [*exec()*](https://www.w3schools.com/python/ref_func_exec.asp) | *Executes the specified code (or object)* |
| [*filter()*](https://www.w3schools.com/python/ref_func_filter.asp) | *Use a filter function to exclude items in an iterable object* |
| [*float()*](https://www.w3schools.com/python/ref_func_float.asp) | *Returns a floating point number* |
| [*format()*](https://www.w3schools.com/python/ref_func_format.asp) | *Formats a specified value* |
| [*frozenset()*](https://www.w3schools.com/python/ref_func_frozenset.asp) | *Returns a frozenset object* |
| [*getattr()*](https://www.w3schools.com/python/ref_func_getattr.asp) | *Returns the value of the specified attribute (property or method)* |
| [*globals()*](https://www.w3schools.com/python/ref_func_globals.asp) | *Returns the current global symbol table as a dictionary* |
| [*hasattr()*](https://www.w3schools.com/python/ref_func_hasattr.asp) | *Returns True if the specified object has the specified attribute (property/method)* |
| *hash()* | *Returns the hash value of a specified object* |
| *help()* | *Executes the built-in help system* |
| [*hex()*](https://www.w3schools.com/python/ref_func_hex.asp) | *Converts a number into a hexadecimal value* |
| [*id()*](https://www.w3schools.com/python/ref_func_id.asp) | *Returns the id of an object* |
| [*input()*](https://www.w3schools.com/python/ref_func_input.asp) | *Allowing user input* |
| [*int()*](https://www.w3schools.com/python/ref_func_int.asp) | *Returns an integer number* |
| [*isinstance()*](https://www.w3schools.com/python/ref_func_isinstance.asp) | *Returns True if a specified object is an instance of a specified object* |
| [*issubclass()*](https://www.w3schools.com/python/ref_func_issubclass.asp) | *Returns True if a specified class is a subclass of a specified object* |
| [*iter()*](https://www.w3schools.com/python/ref_func_iter.asp) | *Returns an iterator object* |
| [*len()*](https://www.w3schools.com/python/ref_func_len.asp) | *Returns the length of an object* |
| [*list()*](https://www.w3schools.com/python/ref_func_list.asp) | *Returns a list* |
| [*locals()*](https://www.w3schools.com/python/ref_func_locals.asp) | *Returns an updated dictionary of the current local symbol table* |
| [*map()*](https://www.w3schools.com/python/ref_func_map.asp) | *Returns the specified iterator with the specified function applied to each item* |
| [*max()*](https://www.w3schools.com/python/ref_func_max.asp) | *Returns the largest item in an iterable* |
| [*memoryview()*](https://www.w3schools.com/python/ref_func_memoryview.asp) | *Returns a memory view object* |
| [*min()*](https://www.w3schools.com/python/ref_func_min.asp) | *Returns the smallest item in an iterable* |
| [*next()*](https://www.w3schools.com/python/ref_func_next.asp) | *Returns the next item in an iterable* |
| [*object()*](https://www.w3schools.com/python/ref_func_object.asp) | *Returns a new object* |
| [*oct()*](https://www.w3schools.com/python/ref_func_oct.asp) | *Converts a number into an octal* |
| [*open()*](https://www.w3schools.com/python/ref_func_open.asp) | *Opens a file and returns a file object* |
| [*ord()*](https://www.w3schools.com/python/ref_func_ord.asp) | *Convert an integer representing the Unicode of the specified character* |
| [*pow()*](https://www.w3schools.com/python/ref_func_pow.asp) | *Returns the value of x to the power of y* |
| [*print()*](https://www.w3schools.com/python/ref_func_print.asp) | *Prints to the standard output device* |
| *property()* | *Gets, sets, deletes a property* |
| [*range()*](https://www.w3schools.com/python/ref_func_range.asp) | *Returns a sequence of numbers, starting from 0 and increments by 1 (by default)* |
| *repr()* | *Returns a readable version of an object* |
| [*reversed()*](https://www.w3schools.com/python/ref_func_reversed.asp) | *Returns a reversed iterator* |
| [*round()*](https://www.w3schools.com/python/ref_func_round.asp) | *Rounds a numbers* |
| [*set()*](https://www.w3schools.com/python/ref_func_set.asp) | *Returns a new set object* |
| [*setattr()*](https://www.w3schools.com/python/ref_func_setattr.asp) | *Sets an attribute (property/method) of an object* |
| [*slice()*](https://www.w3schools.com/python/ref_func_slice.asp) | *Returns a slice object* |
| [*sorted()*](https://www.w3schools.com/python/ref_func_sorted.asp) | *Returns a sorted list* |
| *@staticmethod()* | *Converts a method into a static method* |
| [*str()*](https://www.w3schools.com/python/ref_func_str.asp) | *Returns a string object* |
| [*sum()*](https://www.w3schools.com/python/ref_func_sum.asp) | *Sums the items of an iterator* |
| [*tuple()*](https://www.w3schools.com/python/ref_func_tuple.asp) | *Returns a tuple* |
| [*type()*](https://www.w3schools.com/python/ref_func_type.asp) | *Returns the type of an object* |
| [*vars()*](https://www.w3schools.com/python/ref_func_vars.asp) | *Returns the \_\_dict\_\_ property of an object* |
| [*zip()*](https://www.w3schools.com/python/ref_func_zip.asp) | *Returns an iterator, from two or more iterators* |

# Date Time

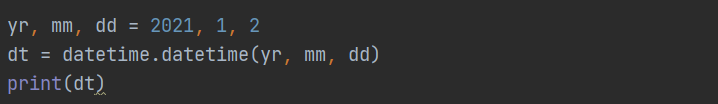
## Current Time:



## Creating Date Object:

To create a date, we can use the datetime() class (constructor) of the datetime module.

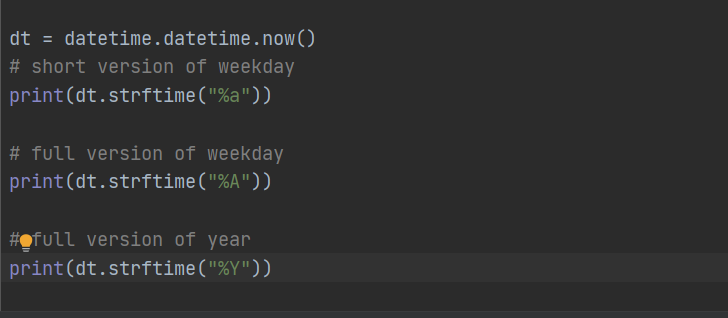
The datetime() class requires three parameters to create a date: year, month, day



## Strftime

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:



Full list of formats:



# Regular Expression

## Raw String:

*A raw string will interpret the string literally.*

***Compile:***

*Compile method allows to separate out pattern in a variable so that we can use that compile later on*

## Interview Questions:

*What is the purpose of PYTHON CASE OK environment variable? What is the purpose of PYTHON HOME environment variable? Is python a case sensitive language?*

*Yes! Python is a case sensitive programming language.*

*What are the supported data types in Python?What is the output of print str if str = 'Hello World!'?*

*It will print complete string. Output would be Hello World!.*

*What is the output of print str[0] if str = 'Hello World!'?*

*It will print first character of the string. Output would be H.*

*What is the output of print str[2:5] if str = 'Hello World!'?*

*It will print characters starting from 3rd to 5th. Output would be llo.*

*What is the output of print str[2:] if str = 'Hello World!'?What is the output of print str \* 2 if str = 'Hello World!'?What is the output of print str + "TEST" if str = 'Hello World!'?What is the output of print list if list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]?*

*It will print concatenated lists. Output would be [ 'abcd', 786 , 2.23, 'john', 70.2 ].*

*What is the output of print list[0] if list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]?*

*It will print first element of the list. Output would be abcd.*

*What is the output of print list[1:3] if list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]?What is the output of print list[2:] if list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]?What is the output of print tinylist \* 2 if tinylist = [123, 'john']?What is the output of print list + tinylist \* 2 if list = [ 'abcd', 786 , 2.23, 'john', 70.2 ] and tinylist = [123, 'john']?What are tuples in Python?*

*A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.*

*What is the difference between tuples and lists in Python?What is the output of print tuple if tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )?*

*It will print complete tuple. Output would be ('abcd', 786, 2.23, 'john', 70.200000000000003).*

*What is the output of print tuple[0] if tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )?What is the output of print tuple[1:3] if tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )?*

*It will print elements starting from 2nd till 3rd. Output would be (786, 2.23).*

*What is the output of print tuple[2:] if tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )?What is the output of print tinytuple \* 2 if tinytuple = (123, 'john')?*

*It will print tuple two times. Output would be (123, 'john', 123, 'john').*

*What is the output of print tuple + tinytuple if tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 ) and tinytuple = (123, 'john')?What are Python's dictionaries?*

*Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.*

*How will you create a dictionary in python?*

*Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).*

*dict = {}*

*dict['one'] = "This is one"*

*dict[2] = "This is two"*

*tinydict = {'name': 'john','code':6734, 'dept': 'sales'}*

*How will you get all the keys from the dictionary?*

*Using dictionary.keys() function, we can get all the keys from the dictionary object.*

*print dict.keys() # Prints all the keys*

*How will you get all the values from the dictionary?*

*Using dictionary.values() function, we can get all the values from the dictionary object.*

*print dict.values() # Prints all the values*

*How will you convert a string to an int in python?*

*int(x [,base]) - Converts x to an integer. base specifies the base if x is a string.*

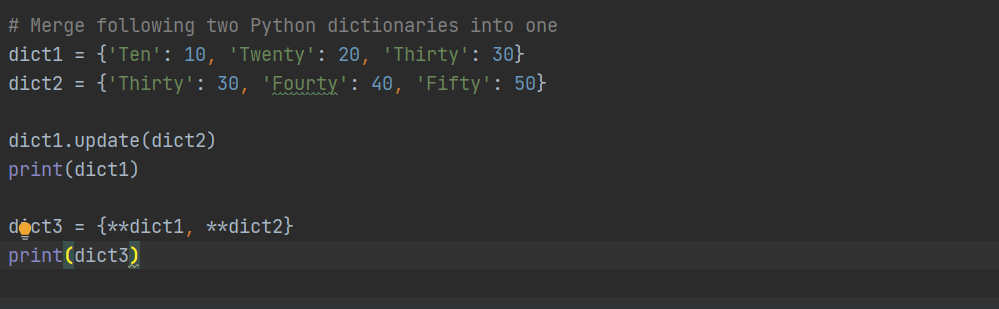
*How will you convert a string to a long in python?*

*long(x [,base] ) - Converts x to a long integer. base specifies the base if x is a string.*

*How will you convert a string to a float in python?*

*float(x) − Converts x to a floating-point number.*

***How to merge two Dictionary?***

******

***How will you convert a object to a string in python?***

*str(x) − Converts object x to a string representation.*

***How will you convert a object to a regular expression in python?***

*repr(x) − Converts object x to an expression string.*

***How will you convert a String to an object in python?***

*eval(str) − Evaluates a string and returns an object.*

***How will you convert a string to a tuple in python?***

*tuple(s) − Converts s to a tuple.*

***How will you convert a string to a list in python?***

*list(s) − Converts s to a list.*

***How will you convert a string to a set in python?***

*set(s) − Converts s to a set.*

***How will you create a dictionary using tuples in python?***

*dict(d) − Creates a dictionary. d must be a sequence of (key,value) tuples.*

***How will you convert a string to a frozen set in python?***

*frozenset(s) − Converts s to a frozen set.*

***How will you convert an integer to a character in python?***

*chr(x) − Converts an integer to a character.*

***How will you convert an integer to an unicode character in python?***

*unichr(x) − Converts an integer to a Unicode character.*

***How will you convert a single character to its integer value in python?***

*ord(x) − Converts a single character to its integer value.*

***How will you convert an integer to hexadecimal string in python?***

*hex(x) − Converts an integer to a hexadecimal string.*

***How will you convert an integer to octal string in python?***

*oct(x) − Converts an integer to an octal string.*

***What is the purpose of \*\* operator?***

*\*\* Exponent − Performs exponential (power) calculation on operators. a\*\*b = 10 to the power 20 if a = 10 and b = 20.*

*What is the purpose of // operator?*

*// Floor Division − The division of operands where the result is the quotient in which the digits after the decimal point are removed.*

***What is the purpose of is operator?***

*is − Evaluates to true if the variables on either side of the operator point to the same object and false otherwise. x is y, here is results in 1 if id(x) equals id(y).*

***What is the purpose of not in operator?***

*not in − Evaluates to true if it does not finds a variable in the specified sequence and false otherwise. x not in y, here not in results in a 1 if x is not a member of sequence y.*

***What is the purpose break statement in python?***

*break statement − Terminates the loop statement and transfers execution to the statement immediately following the loop.*

***What is the purpose continue statement in python?***

*continue statement − Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.*

***What is the purpose pass statement in python?***

*pass statement − The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.*

*How can you pick a random item from a list or tuple?*

*choice(seq) − Returns a random item from a list, tuple, or string.*

*How can you pick a random item from a range?*

*randrange ([start,] stop [,step]) − returns a randomly selected element from range(start, stop, step).*

*How can you get a random number in python?*

*random() − returns a random float r, such that 0 is less than or equal to r and r is less than 1.*

*How will you set the starting value in generating random numbers?*

*seed([x]) − Sets the integer starting value used in generating random numbers. Call this function before calling any other random module function. Returns None.*

*How will you randomizes the items of a list in place?*

*shuffle(lst) − Randomizes the items of a list in place. Returns None.*

*How will you capitalizes first letter of string?*

*capitalize() − Capitalizes first letter of string.*

*How will you check in a string that all characters are alphanumeric?*

*isalnum() − Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.*

*How will you check in a string that all characters are digits?*

*isdigit() − Returns true if string contains only digits and false otherwise.*

*How will you check in a string that all characters are in lowercase?*

*islower() − Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.*

*How will you check in a string that all characters are numerics?*

*isnumeric() − Returns true if a unicode string contains only numeric characters and false otherwise.*

*How will you check in a string that all characters are whitespaces?*

*isspace() − Returns true if string contains only whitespace characters and false otherwise.*

*How will you check in a string that it is properly titlecased?How will you check in a string that all characters are in uppercase?How will you merge elements in a sequence?*

*join(seq) − Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.*

*How will you get the length of the string?*

*len(string) − Returns the length of the string.*

*How will you get a space-padded string with the original string left-justified to a total of width columns?*

*ljust(width[, fillchar]) − Returns a space-padded string with the original string left-justified to a total of width columns.*

*How will you convert a string to all lowercase?*

*lower() − Converts all uppercase letters in string to lowercase.*

*How will you remove all leading whitespace in string?*

*lstrip() − Removes all leading whitespace in string.*

*How will you get the max alphabetical character from the string?*

*max(str) − Returns the max alphabetical character from the string str.*

*How will you get the min alphabetical character from the string?*

*min(str) − Returns the min alphabetical character from the string str.*

*How will you replaces all occurrences of old substring in string with new string?*

*replace(old, new [, max]) − Replaces all occurrences of old in string with new or at most max occurrences if max given.*

*How will you remove all leading and trailing whitespace in string?*

*strip([chars]) − Performs both lstrip() and rstrip() on string.*

*How will you change case for all letters in string?*

*swapcase() − Inverts case for all letters in string.*

*How will you get titlecased version of string?*

*title() − Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.*

*How will you convert a string to all uppercase?*

*upper() − Converts all lowercase letters in string to uppercase.*

*How will you check in a string that all characters are decimal?*

*isdecimal() − Returns true if a unicode string contains only decimal characters and false otherwise.*

*What is the difference between del() and remove() methods of list?*

*To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method if you do not know.*

*What is the output of len([1, 2, 3])?*

*3.*

*What is the output of [1, 2, 3] + [4, 5, 6]?*

*[1, 2, 3, 4, 5, 6]*

*What is the output of ['Hi!'] \* 4?*

*['Hi!', 'Hi!', 'Hi!', 'Hi!']*

*What is the output of 3 in [1, 2, 3]?*

*True*

*What is the output of for x in [1, 2, 3]: print x?*

*1 2 3*

*What is the output of L[2] if L = [1,2,3]?*

*3, Offsets start at zero.*

*What is the output of L[-2] if L = [1,2,3]?*

*L[-1] = 3, L[-2]=2, L[-3]=1*

*What is the output of L[1:] if L = [1,2,3]?*

*2, 3, Slicing fetches sections.*

*How will you compare two lists?*

*cmp(list1, list2) − Compares elements of both lists.*

*How will you get the length of a list?*

*len(list) − Gives the total length of the list.*

*How will you get the max valued item of a list?*

*max(list) − Returns item from the list with max value.*

*How will you get the min valued item of a list?*

*min(list) − Returns item from the list with min value.*

*How will you get the index of an object in a list?*

*list.index(obj) − Returns the lowest index in list that obj appears.*

*How will you insert an object at given index in a list?*

*list.insert(index, obj) − Inserts object obj into list at offset index.*

*How will you remove last object from a list?*

*list.pop(obj=list[-1]) − Removes and returns last object or obj from list.*

*How will you remove an object from a list?*

*list.remove(obj) − Removes object obj from list.*

*How will you reverse a list?*

*list.reverse() − Reverses objects of list in place.*

*How will you sort a list?*

*list.sort([func]) − Sorts objects of list, use compare func if given.*