Python is a dynamic programming language.

**Languages**

🡪 compiled languages – convert to low level script & then be executed

Errors in compiled languages – compile time errors & runtime errors

🡪 Interpreter Language – would execute line by line

Python is both compiled & interpreted language.

Every python file(.py file) when executed for the 1st time it generates .pyc file, it has all object declaration statements.

For every object in python(dir(any attribute/variable))would have ones that have \_\_(double underscores) are magic methods/dunders & attributes

These are not directly envoked by user but it happens internally from class of certain action

**Magic methods are prune to operator over loading whereas parameters are not.**

When we do

a+10 🡪 backend 🡪 a+\_\_add\_\_(10)

**Datatypes**

1.int – object

Whereabouts of python?

1. **Id –** memory storage, python is a heap storage infact all dynamic programming languages use heap storage also all dynamic attributes of any language

If a variable is defined with a number then that num is no where assigned in same session, if same number is assigned again to different variable instead of assigning new memory it js reference to same memory location

If we r assigning a different number to some variable which is assigned, the memory & its associated value is not deleted but a new memory with new value with reference to the variable

By the end of the session, global interpreter lock(GIL) is would clear all the memory locations with associated value which are not referencing by any variables

1. **Type()**
2. **Dir()**
3. **Help()**

**Differences between python 2.x & python 3.x**

1. Change in print syntax, in python 2.x parenthesis is not mandatory where as it’s mandatory in python 3.x
2. In 2.x, strings are stored as ASCII & in 3.x they are stored as Unicode
3. In 2.x, integer division rounds to nearest whole number & in 3.x it’s results in exact result(float)
4. Python 3 is backward incompatible
5. Xrange in python 2.x is replaced with range function & performs btr while iterating over sequences.
6. 2.x More complex syntax, 3.x simpler & easy to understand
7. Slower performance in python 2.x was improved to 3.x

**What is ASCII ?**

(American standard code for information interchange)

It’s a character encoding standard used for communication. Every alphabet, number & special characters have a unique code.

**Unicode**

Describes how characters are represented by code points. Code point value ranges from 0 to 0x10FFFF. UTF-8 is an encoding system for Unicode. Unicode is understandable & used across different platforms & applications.

**Methods in Pyhton**

1. Class Methods

It takes cls parameter hence can change only class state but not object state directly. However, class methods can still modify class state that applies across all instances of the class

1. Instance methods

It takes self parameter which refers to object & hence can change object state

Instance methods can access class also by self.\_\_class\_\_ attribute & hence can change class state. Along with self we can include other parameters as well in instance methods.

1. Static methods – they don’t have self or cls parameters hence they can’t change neither cls or object state. Static methods are restricted in what data they can access.
2. class MyClass:
3. def method(self):
4. return 'instance method called', self
5. @classmethod
6. def classmethod(cls):
7. return 'class method called', cls
8. @staticmethod
9. def staticmethod():
10. return 'static method called'

**self Parameter**

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class

**what is short integer caching in python**

Python caches small integers ranging between -5 to 256.

These numbers are used frequently so these integers will be assigned at startup.

Then each time a variable is referenced, we would be referencing an object which is already allocated, hence it's better for performance.

**What is metaclass in python.**

Metaclass is a class that defines how class behaves. A class itself is instance of metaclass

C is the basis of python ->Cpython

Java basis of python ->Jython

.net basis of python -> iron python

**Why do we have multiple implementations/definitions of strings?**

This flexibility is designed to make it easier for developers to write strings without having to escape the quote character.

**Python Array**

Publicly dynamic in nature

non-continuous in storage

follows indexing concept

follows slicing concept

traversal is always from left to right

array[start\_index:stop\_index:step\_index]

supports negative indexing

index starts at zero

strings in python are implemented as arrays

slicing concepts handles exceptions

a = “hello”

a[:100] – displays ‘hello’ as slicing can handle exceptions

a[5:2] – returns ‘’ i.e., null string as traversal is always from left to right

a[-1:-3] - returns ‘’ i.e., null string as traversal is always from left to right

a[-3:-1] – returns ‘ll’ – end position is not inclusive

a[::100] – returns ‘h’ it skips 100 elements

a[::-1] – returns reverse of provided string ‘olleh’

a[1:5: -1] – returns ‘’ (null string)

a = “hello world”

a[-1:-5:-1] – returns ‘dlro’

**why do index start at 0?**

array indexing start with 0 as array\_name by itself can be used as the memory address in this case else if it starts with 1 to find the memory location we need to add 1 to the memory location

**String formatting**

Name = pavani

Age = 24

Print(“I am pavani & I’m 24”)

Print(“I am”+name+” & I’m “+str(age))

Print(f‘I am {name} & I’m {age}’)

Print(“I am {} and I’m {}”, format(name, age)) – here {} are called placeholders

Print(“I am {1} and I’m {2}”, format(name, age))

**Find**

S = ‘I am harry’

S.find(‘a’) 🡪 2 i.e., first occurrence of ‘a’ in the string

s.find(‘a’,3)🡪 6 i.e., first occurrence of ‘a’ as it starts search 3rd position

syntax: s.find(element, start, stop) start being inclusive while stop being exclusive

s.find(‘a’,3,6) 🡪returns -1 meaning it’s a null string

s.count(‘a’) 🡪 returns 2 as it displays number of occurrences of a character

a = ‘I am harry and I am software and I am a software engineer. I love long drives.’

##second occurrence

First\_occurence = a.find(‘a’)

a.find(‘a’, First\_occurence+1)

(or)

a.find(‘a’, a.find(‘a’)+1)

**Convert a string to list**

Split is used to convert string to list

s.split(‘ ‘)

**convert list to string**

join

‘ ’.join(a)

🡪 we can convert string to list & list to string by type casting but it unpacks every character

**In & not in**

‘te’ in ‘harry’ 🡪 false

Note: It’s case sensitive

‘Ha’ in ‘harry’ 🡪 false

Strip – removes whitespaces or selected characters.

lstrip/rstrip – same as strip, removes whitespaces on left & right respectively

endswith()

startswith()

upper()

capitalize()

**escape characters - \**

**List – immutable**

Is by default a dynamic array (no specification of length). Dynamic structures are always stored using Heap/stack

A = arr[10] -> even if there are no values it still occupies 10 elements storage

* Index, slicing applicable

Basic types are datatypes & complex types are containers.

Based on scenario, we use different containers considering time & space complexity

**Swapping**

**Sorting –** a.sort() changes the original list whereas sorted(a) results sorted list but original list is not changed.

**Difference between append vs insert vs extend**

Append adds element to the end of the string

Extend also adds the elements of the iterable to the end of the list but it unpacks & add them. Also it takes only iterable

A = [1,2,3]

A.append(‘43’) 🡪 A = [1,2,3,’43’]

A.extend(‘43’) 🡪 A = [1,2,3,4,3]

**Difference between pop & remove**

Pop uses index to remove the element & returns the deleted value whereas remove takes the element to be removed as input & deletes the 1st occurrence of that element in the list

**index**

a.index(‘r’) – gives the index of that element in a.

data = [“hary”,’entrepreneur’, ‘Miami’, 10000]

print(f’my name is {data[0]} & I am an {data[1]}’)

Hashing – effective way of storage in both time & space complexities, we use hash function to find keys which would point to the memory location where the elements are stored.

**Find vs rfind**

Find returns first occurrence of a string where as rfind returns last occurrence

**Index vs rindex**

Index returns first occurrence of a string where as rindex returns last occurrence

**List**

* What is difference between deep copy & shallow copy
* difference between list & tuple

tuple is faster than list

avoid using list as much as possible

we use tuples for iterations that restrict data change like iterating source data

tuple will freeze the index of elements once declared

**Sets**

2 properties – every element of domain should be mapped

Domain should not be multi-mapped

|  |
| --- |
| can store only immutable elements |
| hashes every elements in set  We cannot include mutable objects in sets as they can’t be hashed  Note: Mutable objects can’t be hashed i.e., every mutable element is non hashable where as every immutable element is hashable |
| doesn't follow order of storage as we use hashing for storage. |
| there is no concept of index in the storage |
| no slicing |
| order changes everytime, Set items can appear in a different order every time you use them |
| order of 1 TC |
| elements should be unique  Once a set is created, you cannot change its items, but you can add new items. |

Stored using heap uses hash function for storaging. Hash function creates a static link between key & value

For an input value, we do hashing & results in a id where value is stored so even sets are key-valued

Heap storage is not continous

**where did you use set in real time projects**

there is one scenario where we are getting source files in json/csv format. you have millions of records in those files. we want to eliminate duplicates in those files so i converted the data records into set object and I removed the duplicates from multiple files

merge two data files without duplicates

to find common elements

**Dictionary**

|  |
| --- |
| key value pair |
| Hashmap for storage |
| keys are hashed |
| keys should be unique |
| values can have duplicates |
| keys cannot have mutable datastrucutes |
| doesn't maintain order by default |
| we can extract value of dictionary using its corresponding key |

Dict eliminates that id’s wnt be readable like in sets where id are storage positions which are huge numbers

Key is pointing the address location of value

If we are trying get value of dict like d[‘key’] if key is not present, it results in error but if we use d.get(‘key’) it returns none if key doesn’t exist

If there are 2 values for a key, the 2nd value would be overwritten by the 1st value

We read json using dict

We deal with multiple file formats like csv, json,excel, xml. We have a standard to convert all these to parquet files in our datalake(reduces the size as it encodes & compresses)

We can use tuple as key but not list. We can use only mutable objects as keys are hashed

d.Items result in list of tuples

d.update takes dict to add elements to existing dict

dict are not stored continuously as its dynamic storage

**why is empty list/set/dict considered as null?**

**Difference between is & ==**

'==' checks if the values are same whereas is checks if points to same memory location

**Cloud - AWS**

**200+ services**

**Public – prepaid/postpaid/bigbased/tenure based**

**Available for public on subscription based**

**Free tier**

**S3 – free – for 5 GB**

**Private –**

**Go to Account->billing & cost management->budgets->create budget**

**-zero spend budget**

**Or customize budget -cost budget– next = give name**

**-expiring budget (for yr ) &**

**Provide mail address**

**& create - alert**