Python is an interpreted language. It does not compile the every line in the code; it only checks the lines in code only when it runs that line in code so even-though a line of code has error in it, program runs fine if that code doesn’t part of execution flow.

Created at 1990, quick and light language. Better suited for automation, considered as scripting language.

def …. : // defining a function

import …. // can refer to external python module

sys.argv // takes the command line arguments, here 0th will be pgm name itself, so start

with 1st argument

**Strings**

Strings are immutable (kind of read only thing, can’t be changed) [35:51]

**Eg :**

a = ‘Hello’;

a.lower(); // This gives ‘hello’, but ‘a’ value is still ‘Hello’

‘%s’ is the place holder for Strings and ‘%d’ is place holder for numbers.

**Eg :**

‘My name is %s and I am %d years old’ % (‘Navnth’, 24)

String indexing will be as below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **H** | **e** | **L** | **L** | **O** |
| **0** | **1** | **2** | **3** | **4** |
| **-5** | **-4** | **-3** | **-2** | **-1** |

**Initialization :**

myString = ‘abcd’; // This is allowed

myString1 = ‘abcd123’; // This is also allowed

myString2 = “sdsd456”; // This is also allowed

myString3 = “I \”love\” this”; // It will print → **I “love” this**

myString + ‘ exercise !!‘ // This prints → **I “love” this exercise!!**

myString[0] → gives ‘**a**’ // first car in string

myString[-1] → gives ‘**d**’ // last char in string

myString[1:3] → gives ‘bc’ //this is slicing, which gives values from index 1 to index 2, but not 3

myString1[2:6] → gives ‘cd12’ // here it gives value from index 2 to index 5, but not 6

myString1[2:] → gives ‘cd123’ // if we don’t mention last slicing point, it will give values till end.

myString[:] → gives ‘abcd123’ // if we don’t mention first and last no, it gives full values

len(myString) → gives **4** // length of string variable ‘myString’

str(122121) // converts anything in to string

Some String methods:

myString.find(‘c’); // It outputs **3**, which is the index of char ‘c’ in myString

‘ffg ugh ighjh’.split(); // this gives the o/p [‘ffg’, ‘ugh’, ‘ighjh’], if no params it consider spaces

**Lists**

Lists are mutable and growable in size.

Eg :

myList = [1, 2, 3] // now myList value is **1, 2, 3**

myList[0] = 13; // now myList value is **13, 2, 3**

List will not yields another list when we do copy a list. [03:01 – day1 part2]

Eg:

myList = [1, 2, 3]

myList1 = myList; // Here, myList1 & myList is pointing same list stored in some location.

myList[0] = 13; // Now values of myList1 and myList are [13, 2, 3].Change in one list affects both lists.

myList1[1] = 33; //Now values of myList1 and myList are [13, 33, 3]

Pythonic way of copying a list.

myList1 = myList[:];

Most of the operations work on Strings work on Lists too.

Eg :

myList = [1, 2, 3 ];

myList + [‘nav’ , ’nith’]; // this will give [1, 2, 3, ‘nav’, ‘nith’ ]

myList == myList1 // this returns **True**

myList[0:2] // this gives [1, 2], slicing syntax same as in strings

This is how we can loop over list elements [ 8:20] [day1, part 2]

***for*** *lst* ***in*** *myList****:*** // this for loop copy every single element of list to lst variable

***Print*** *lst****;*** // this prints every single list elements when it loops in to the list

2 ***in*** myList **;** // This returns True; this is way we can check if elements exist in list

5 ***in*** myList**;** // This returns False;

Some of built-in functions:

append();

myList.append(nav); // This modifies the list myList as [1, 2, 3, ‘nav’]

myList1 = myList.append(nav); // this can be wrong as append() returns nothing

***myList = [1, 2, 3, 4];***

***myNewList=[];*** // initialize new empty list

***for n in myList:***

***myNewList.append(n);*** // copying all myList values in to myNewList list.

pop();

myList.pop(0); // This removes 0th elements from list myList

del();

del myList; // delete the whole list and removes the reference for variable myList

del myList[0]; // This gives values [2, 3], as it removed 0th element 1

***a = [1, 4, 3, 2]***

***b=a;***

***del a;*** // In this case **b** still has values [1,2,3]

sorted();

sorted(a); // this gives the o/p of [1, 2, 3, 4] as new list, so can assign o/p as below

b = sorted(a); // this copies sorted ‘a’ list to ‘b’ vaiable

sorted(a, reverse = True); // this gives the o/p of [4, 3, 2, 1]

b = [‘dd’, ‘bbbb’, ‘aaa’, ‘c’] // create a list with each argument has multiple chars

sorted(b, key = len) // this gives o/p as [‘c’, ‘dd’, ‘aaa’, ‘bbbb’], based on the length

c = [‘dd’, ‘bbbb’, ‘aaz’, ‘c’]

sorted(c, key = Last) //this gives o/p as [‘bbbb’, ‘c’, ‘dd’, ‘aaaz’], bases on Last char

‘:’.join(c) // this gives o/p as ‘**dd:bbbb:aaz:c**’

Now value of ‘c’ is ‘dd:bbbb:aaz:c’

c.split(‘:’) // this gives o/p as its original form [‘dd’, ‘bbbb’, ‘aaz’, ‘c’]

**range(int) ;** // kind of readymade list provide values up to range given

range(5); //gives values as [ 0, 1, 2 , 3, 4], we can do all list operations.

**Tuple**

Tuples are fixed in size and are immutable(values can’t be changed) (32:00)

a = (1, 4, 2, 3); // initializing tuples

x, y = 1, 2 // this will assign **x** to **1** and **y** to **2**

b = [ (1, ”b”), (3, “a”), (1, “a”) ] // creating list of tuples

All operations we do on Strings and Lists can be done on Tuples too.

len(a); // this gives 4

sorted(a); // this gives 1, 2, 3, 4

sorted(b); // this gives [(1, ‘a’), (1, ‘b’), (3, ‘a’)]

tuples = [(1, 3), (3, 2), (2, 1)];

sorted(tuples, key=operator.itemgetter(-1)); // this will sort based on last element in tuple.

sorted(tuples, key = lambda x: x[-1]); // this will also sort based on last element in tuple

**Control Statements**

**If :**

if **a == b :** // No brackets required.

if **a==b** and **b==c :** // we can use ‘and’, ‘or’, ‘not’ words in **if** for logical comparison

**Dictionary**

Very fast data structure, maps data with keys so that retrieval would be faster.

Can be used to map random piece of data with other occurrences of same random data.

Eg : Can be used to map different ip address to check how many times different servers are hitting ip by reading log file.

Google uses this to map url for every word user typed in search box.

d{ } // this is how we initialize a dictionary

d[‘a’]= ‘abchdef’; // here we adding ‘abchdef’ is data to ‘a’ is key for dictionary ‘d’

d[b’] = ‘xyzabchd’;

d.get(‘a’); // gives the values of key ‘a’

‘a’ in d : // this returns **true** if ‘a’ key present in dictionary ‘d’

d.keys(); // returns all the keys present in dictionary ‘d’

d.values(); // gives values of all the keys

sorted(a.keys()); // gives all the keys in sorted fashion

d.items(); // gives list of tuples of all the keys and their values of dictionary ‘d’

// Eg : [(‘a’, ‘abchdef’) , (‘b’, ‘xyzabchd’)]

Some other operations with dictionary:

***for*** *k* ***in*** *d.keys():*

***print*** *‘key: ‘, k , ‘→’,d[k] ;* // key: a → ’abchdef’ key: b → ‘xyzabchd’

***for*** *tuple* ***in*** *d.items()****:***

***print*** *tuple****;*** // (‘a’, ‘abchdef’) (‘b’, ‘xyzabchd’)

**Files**

Simple pgm to read contents of a file.

**First method:** Reading line by line, saves ram memory.

Eg: If you are 40GB file and it may not that memory to read that as it reads a single line at a time.

def Cat(filename):

f = open(filename, ‘r’); // open() opens a file and it will take parameters

for line in f: // reads file contents line by line

print line;

def main():

Cat(filename);

o/p : will come in line by line like below

…..

…..

**Second method:** Reading entire file as list.

Eg: If you want to read entire file as a list, you can use this way but at the cost of ram memory.

def Cat(filename):

f = open(filename, ‘r’); // open() opens a file and it will take parameters

lines = readlines(); // reads entire file contents a list

print lines;

def main():

Cat(filename);

o/p : will come as list like below

[‘…..’ , ‘…..’ ]

**Third Method:** Reading entire file as a string.

Eg: if you want to read a file and want to perform some operations(regex operations) with contents of file then you can use below method to read entire file content as string.

def Cat(filename):

f = open(filename, ‘r’); // open() opens a file and it will take parameters

text = read(); // reads entire file contents as a single string.

print text;

def main():

Cat(filename);

o/p : will come as string like below

…..

…..

**Regular Expressions**

‘re’ module supports REGEX in python.

re. search( pattern, string ); // this is the method we use for REGEX in python, returns match obj

match.group(); // shows the matching text if pattern exist, else display an error

Eg:

match = re.search(‘ th’ , ‘navanith’);

match.group(); // prints ‘th’, which is matched text.

Here if there is any pattern exists then it returns something to ‘match’ if pattern doesn’t exist then it will not return anything and pattern should be matched 100% with the string given.

**Simplest pgm to pattern matching:**

***def*** *Find (pattern, string):*

*match = re.****search****(pattern, string);*

***if*** *match:*

***print*** *match.****group****();*

***else****:*

***print*** *‘not found’;*

There are 2 rules in REGEX:

1. Simple chars match themselves.

2. Special chars we need to follow below table. // this will traverse left to right

**. (dot) matches any char including spaces , except new line.**

Eg :

re.search(‘. .h’ , ‘navanith’); // this will give ‘ith’

re.search(‘n..h’, ‘navanith’); // this will give ‘nith’

re.search(‘..h’, ‘navanegh navaiith’); //this will give first ‘egh’, if u want 2nd ‘h’ give pattern ‘i.h’

re.search(‘\.c’, ‘nac.chg’); // this will give ‘.c’ , use ‘backslash .’ to match a dot.

re.search(**r**‘\.c’, ‘nac.chg’); // here if we specify ‘r’ , it takes raw pattern – does not process spcl chrs

**‘\w’ matches word chars – only letters, digits, underscore.**

Eg:

re.search(r ‘:\w\w\w’ , ‘blah :bla blah); // this will give ‘:bla’

**‘\d’ matches only digits.**

Eg:

re.search(r ‘\d\d\d’ , ‘blah :123abcblah); // this will give ‘123’

**‘\s’ matches only white space including tabs and new line.**

Eg:

re.search(r ‘\d\s\d\s\d’, ‘blah :1 2 3abcblah); // this will give ‘1 2 3’

**‘\S’ matches all non white space including special chars. and new line.**

Eg:

re.search(r ‘:\S+’, ‘bl ah:1&2+123hfhfhg,mk 3abcblah); // this will give ‘:1&2+123hfhfhg,mk’

**The below 2 are somewhat important in REGEX: --- [ 20:00][D2 P1]**

**‘+’** 1 or more

**‘\*‘** 0 or more

Eg:

**re.search(r ‘\d\s+\d\s+\d’, ‘abch1 2 3 djnjmm);**

// above will matches with 1 or more spaces b/w the digits.

**re.search(r ‘\d+’, ‘abch123djnjmm);**

// this displays ‘123’, here we can see ‘+’ display as long as it matches and stops where it is not.

**Square bracket ‘ [] ’**

These square brackets can be used to indicate set of characters. If you put chars with-in square bracket ‘[ ]’, that indicate only those kind of chars should be allowed.

Eg :

**re.search(r’[\w.]+@[\w.]+’, ‘sjhdsksds** [**navanith.hv@gmail.com**](mailto:navanith.hv@gmail.com) **shdgsds dfhdjf’);**

**or**

**re.search(r’\S+@\S+’, ‘sjhdsksds** [**navanith.hv@gmail.com**](mailto:navanith.hv@gmail.com) **shdgsds dfhdjf’);**

// Above 2 will matches the email id properly. Here if we write ‘.’(dot) inside square bracket means it is a dot, not any matching char and ‘\w’ will be processed as matching char, as usual.

But, above pattern will not work if email address is in ‘.[navanith.hv@gmail.com](mailto:navanith.hv@gmail.com)’ format. In this case we have to use below pattern

**re.search(r’\w[\w.]\*@[\w.]+’, ‘sjhdsksds .**[**navanith.hv@gmail.com**](mailto:navanith.hv@gmail.com) **shdgsds dfhdjf’);**

**Parenthesis ‘( )’ ---- [27:20][d2 p1]**

These parentheses can be used to define set of cars.

Eg :

**m = re.search(r’([\w.]+)@([\w.]+)’, ‘sjhdsksds** [**navanith.hv@gmail.com**](mailto:navanith.hv@gmail.com) **shdgsds dfhdjf’);**

m.group(); // this will give ‘[navanith.hv@gmail.com](mailto:navanith.hv@gmail.com)’

m.group(1); //it give the value of 1st most parenthesis which is ‘navanith.hv’, the user name

We can put ‘+’ inside/outside of parenthesis.

**findall()** method :

**re.findall(r’[\w.]+@[\w.]+’, ‘sjhdsksds** [**navanith.hv@gmail.com**](mailto:navanith.hv@gmail.com) **shdgsds dfhdj gfhshsj@hshjsjk f’);**

// this will give all occurrences of matched pattern as list of strings

**[‘**[**navanith.hv@gmail.com**](mailto:navanith.hv@gmail.com)**’, ‘gfhshsj@hshjsjk’ ]**

**m = re. findall (r’([\w.]+)@([\w.]+)’, ‘sds** [**navanith.hv@gmail.com**](mailto:navanith.hv@gmail.com) **shdgsds dfhdj gfhshsj@hshjsjk f’);**

// this will give all occurrences of matched pattern as list of tuples

**[(‘**[**navanith.hv’, ‘gmail.com**](mailto:navanith.hv@gmail.com)**’ ),( ‘gfhshsj’ , ‘hshjsjk’) ]**

give ***dir(re)*** to see the list of parameter allowed ‘re’ module, below is one such example.

**re.findall(r’[\w.]+@[\w.]+’, ‘sjhds** [**nav.hv@gmail.com**](mailto:nav.hv@gmail.com) **shds dfhdj gfhsj@hshjsjk f’, re.IGNORECASE);**

**os(operating system) module**

os.listdir(path); // takes path in string format and gives directory names in form of list

os.path(); // it has utilities to manipulate file path

p = os.path.join(directory, filename); // joins the directory to the filename

os.path.abspath(p); // gives the absolute path name of the all filenames in that directory

os.path.exists(‘/e:/learn’); // returns true if that path exists, else false

os.mkdir(‘/e:/xyz’); // creates xyz dir in the specified path

**Some other helpful utilities in diff modules**

shutil.copy(source, destination); //copy anything

**commands** // this module can be used to launch external process and to finish the same

**Exceptions**

***try***:

………

………

***except*** exceptionName:

………

………