Friday
28/02/2020

Q1. Explain python's simplest builtin data types?

Ans: The pytho's simplest build-in data types are

- boolean
- integers
- float
- Strings.

Boolean :-

The boolean type is either true or false In python, boolean variables are defined by the True and False keyword.

Eg: ① a = True
b = False
print (a)

Some of the boolean operator in python are

- or
- and
- not
- == (equivalent)
- != (not equivalent)

Summary of boolean arithmetic and boolean operators is shown

A	B	not A	not B	A==B	A!=B	A or B	A and B
T	F	F	T	F	T	T	F
F	T	T	F	F	T	T	F
T	T	F	F	T	F	T	T
F	F	T	T	T	F	F	F

Eg: A = True

B = False

>> A or B

Integers :-

Integer is a whole number such as 42 and 10000000. In python there is effectively no limit to how long an integer value can be. of course it is constrained by the amount of memory of system.

Any sequence of digits in Python is assumed to be a literal integer.

>>> 5

>>> 0.

The negative and positive integers in Python can be specified by prefix sign

iv) Extract a character with [ ]:

To get a single character from a string specify its offset inside square brackets after the string name. The first (leftmost) offset is 0, the next is 1 and so on. The last (rightmost) offset can be specified with -1 ... -2, -3 and soon.

Eg: letters = 'abcdefghij'

```
letters[0]
'a'
letters[1]
'b'
letters[-1]
'j'
```

v) Slice with [start : end : step]

In python, substring can be extracted from a string by using a slice. Slice are define by square brackets, a start offset, end offset and an optional step size. The slice include characters from offset start to one before end.

- [:] extracts the entire sequence from start to end.

Eg: letters = 'abcdefg'

letters[:]
'abcdefg'

•) [start:] ⇒ specifies from the start to end

Eg:-
letters[2:]
'cdefg'

•) [:end] specifies from the begining to end minus 1.

Eg:-
letters[:2]

•) [start:end] indicates from the start to end minus 1.

Eg: letters[2:5]
'cde'

•) [start:end:step] extracts from the start to end minus 1 skipping characters by steps

Eg: letters[2:6:2]
'ce'

06. Explain the following string functions ?
(a) len() (b) join() (c) split()

Ans:- a) len() :

The len() function counts characters in the string !

Eg: >> len (letters)
→ 25
>>> empty = " "
>>> len (empty)
→ 0

b) Split with split() :

built in string split() function is use to break a string into a list of smaller strings based on some separator

Eg: todos = 'get gloves, get mask, give cat vitamins, call ambulance'
>>> todos. split (',')
→ ['get gloves', 'get mask', ' give cat vitamins', ' call ambulance']

In these example the .split function is called with argumen (',') which splited

the entire string whenever ';' is encountered. If no argument is specified then the output would be like.

```
>>> todos.split()
→['get', 'gloves, get', 'mask, give', 'cat',
  'vitamins, call', 'ambulance'].
```

(c) Combine with join()

join() function is just opposite to split() function. The join() function collapses a list of strings to a single string with string.join(list). To join the list lines with separating newlines '\n'.join(lines) can be used.

```
Eg: crypto_list = ['yeti', 'Bigfoot', 'loch']
    crypto_string = ';'.join(crypto_list)
    print('Found a singing book deals:',
          crypto_string)
```

→ Found and singing book deals:
yeti; Bigfoot; loch.

Q1. Write a small program to receive a string, input, find the length, change the case (upper u to l / l to u), center the text, left justify, right justify substitute with replace.

Ans:

```
Str1 = string (input ('Enter a string:'))
len (str1)
Str1.upper()
Str1.lower()
Str1.center(30)
Str1.ljust(30)
Str1.rjust(30)
Str1.replace('hello', 'hi')
```

Q8. Name the data structures in python ?

Ans: The python offers 4 data structure namely.

*) Lists :

The list are the ~~not~~ hetrogenous collection of items. The list are mutable. They can be created by square bracket [].

Eg: empty_list = []

week_list = ['sun', 'mon', 'tue', 'wed' ...]

*) Tuples :-

Tuples are the sequence of arbitrary items. Tuples are immutable. They can be created by paranthesis

Eg: empty_tuple = ()

M tuple = ('Groucho', 'chicho', 'Harpo')

*) Dictionaries :-

A dictionary is similar to a list but items are not selected by offset instead specify by a unique value they to associate with each value means the items in dictionary exists in the

form (key: value) pair. They are created by {} brackets.

Eg: empty_dict = {}

dict_eg = {"day": "Sunday", "day2": "Monday"}

*) Sets :-

A set is like a dictionary with its values thrown away, leaving only the keys. As with a dictionary each key must be unique. Sets are used ~~on~~ when ~~only~~ you want to know that something exists and nothing else ~~does~~ about it.

Eg: even = {0, 2, 4, 6, 8}

---

09. Briefly explain list with example.

Ans List :-

List is a orderly collection of hetrogeneous and mutable data item. The list can be created with zero or more elements, seperated by commas and surrounded by square brackets.

Eg: empty_list = [ ]

Eg: another_empty_list = list()
>>> another_empty_list
→ []

*) convert other data types to lists with list()
Eg: list ('cat')
['c', 'a', 't']

*) Split a list.
Eg: split_bday = '1/6/1952'
split_bday.split('/')
['1', '6', '1952']

*) Get an item by using [offset]
Eg: fruits = ['mango', 'apple', 'guva']
>>> fruits[0]
'mango'
>>> fruits[-1]
'guva'

*) lists of lists:
Eg: fruits = ['mango', 'grapes']
veg = ['tomato', 'potato']
food = ['fruits', pie veg]

*) Change an item by [offset]
Eg: fruits = ['apple', 'mango', 'cherry', 'grapes']
fruits[2] = 'strawberry'
→ ['apple', 'mango', 'strawberry', 'grapes']

*) Slice to extract items by offset Range.
Eg: fruits[0:2]
['apple', 'mango']

*) Add an item to end with append()
Eg: fruits.append['kiwi']
['apple', 'mango', 'strawberry', 'grapes']

*) Add by offset with insert()
Eg: fruits.insert(3, 'Gava')
fruits
['apple', 'mango', 'strawberry', 'Gava']

*) Delete by offset with del
Eg: del fruits[-1]
fruits
['apple', 'mango', 'strawberry',]

*) Delete by remove()
Eg: fruits.remove('strawberry')
fruits
['apple', 'mango']

*) Delete by pop()

Eg: fruits.pop()
'mango'

Eg: fruits = ['apple', 'mango', 'grapes']
fruits.pop(1)
'mango'

10. Explain tuple data structure with example?

Ans: Tuples :-

Similar to lists, tuple are sequence of arbitrary items. Unlike lists, tuples are immutable, meaning you can't add, delete, or change items after the tuple is defined. So a tuple is similar to a constant list.

The syntax to make tuples is a little inconsistent show below

```
>>> empty_tuple = ()
empty_tuple
()
```

(+, -)

Eg: >> +123
>>> -123.

Float :-

Integers are whole numbers, but floating-point numbers (called floats in python) have decimal points. floats are handled similarly to integers. The operator like +, -, *, /, //, **, and % can also be used and divmod() function

Eg: >>> float (True)
>>> float ('98.6')
>>> float ('-1.5')

Strings :-

Strings are sequence of character data. The String type in python is called str. String in python are immutable. you can't change a string in place. but you can copy parts of strings to another string to get same effect.

Eg: >>> 'Snap'
>>> "crackle"

To make a tuple with one or more element follow each element with a comma.

Eg: one_max = 'Groucho'
one_max
('Groucho',)

Eg: max_tuple = ('Groucho', 'chico', 'Harpo')
a, b, c = max_tuple
>> a
'Groucho'
>>> b
Chicho.

This is sometimes called tuple unpacking.

Eg: password = 'Swift'
icecream = 'tutti fruitti'
password, icecream = icecream, password.
>>> password.
'tutti fruitti'
>>> icecream
'Swift'

11) Differentiate lists and tuple

Ans: List	Tuples.
*) List use more space	Tuples use less space
*) Lists are mutable	Tuple are immutable
*) Implication of iterations is Time consuming.	Implication of iterations is comparatively faster.
*) Lists is better for performing operations such as ~~act~~ insertion and deletion	Tuple data type is appropriate for accessing the elements
*) Lists have several built in methods	Tuples does not have must builtin method
*) The unexpected changes and errors are more likely to occur	In tuple it is hard to take place.

*) List is defined using Square brackets []	Tuples are defined using parentheses ()
*) They can't use as dictionary key	Tuples can be used as dictionary key
*) Eg: empty list=[]	Eg: Empty tuple=()

Q.12. Write a program to a list and do the following operations

i) change an item by offset
ii) Add an item to end of existing list
iii) Combine two lists
iv) Delete an item by offset
v) Delete an item by value.

⇒

```
languages=['Python','C','C++','Java']
print(languages)
languages[2]='CPP'
print(languages)
languages.append('C#')
print(languages)
web_lang=['Html','xml','php']
languages.extend(web_lang)
```

```
print (languages)
del language [-1]
print (languages)
language.remove ('c++').
print (languages)
```

13. Write a program to create a tuple and show each item from tuple.

→

```
languages = ('PHP', 'Python', 'C', 'C#')
print (languages)
print (languages[0])
print (languages[-1])
print (language[:3])
web_lang = ('php', 'Html', 'xml')
print (web_lang)
All_lang = ('cobol', ('c', 'c++', 'java'))
print (All_lang)
a, b, c, d = web_lang.
print (a)
print (b)
print (c)
```

Monday
02/03/2020

Collect minimum 25 functions associated with list in python.

	function	Description
1)	clear()	Removes all the elements from the list
2)	copy()	Returns a copy of the list
3)	count()	Returns the number of elements with specified value
4)	extend()	Adds the elements of a list to end of current list
5)	index()	Returns the index of first element with the specified value
6)	insert()	Adds an element at specified position
7)	pop()	Remove the element at the specified position
8)	remove()	Removes the first item with the specified value

9) reverse()	Reverse the order of list
10) sort()	sorts the list
11) append()	Adds an element at the end of the list
12) del()	delete the element of list with index
13) ~~count~~ sum()	calculates sum of all elements of list
14) min()	calculates minimum of all Elements of list
15) max()	calculate maximum of all elements of list
16) get()	Returns a ~~copy~~ value of Specified key
17) items()	Returns a list containing a tuple for each key value pair

18)	len()	Gives the total lengths of the list
19)	cmp ()	compare elements of both list
20)	list (seq)	Converts a tuple into list
21)	Split ()	Split the list into a list by some separator string
22)	list ()	create a another list
23)	join()	combine's the list to make a single list
24)	range (stop)	Represents an immutable sequence of no's & is commonly used for looping a specific number of time in for loops
25	in	to check the value is existing in the list or not.

03/03/2020

Page No. 28
Date

1. Write a program in python to create a dictionary to store author and author name as pair value and access them with key value 'author'

```
dict1 = {'author1' : 'Kvkk prasad',
         'author2' : 'Bill lubanovic',
         'author 3' : 'Kelvin',
         'author 4' : 'William Stalling',
         'author5' : 'Prakash Yeri'}
dict1.keys()
dict1.get('author 1')
list(dict1.values())
list(dict1.items())
```

Page No. 29
Date

2. Create a list of title, author, year of publication, then convert the list to dictionary and access the value using any key

```
book = [['title', 'Python'], ['author', 'Bill L'], ['year of P', '1998']]
dictbook(book)
dictbook.get('title')
dictbook.get('title', 'author')
dictbook.keys()
```

3. Develop / create a dictonary for climate data for sharing among any wheather forecasting stations and implement all basic dictonary operations.

Q2. Explain variables, names and objects in python?

Ans: Variables, Names and Objects :-

In python everything - booleans, integers, floats, strings, even large data structures, function and programs - is implemented as an object. This gives the language a consistency that some other languages lack. An object is like a clean plastic box that contains data. The object has a type such as boolean or integer that determines what can be done with data. The type also determines if the data value contained by box can be changed (mutable) or immutable (constant).

The python is strongly typed which means that the type of an object does not change even if its value is mutable.

Programming language allow you to define varibles. These are names that refers to values in the computer memory that you can refer to use with program. '=' is used to assign a value to a variable.

Eg: a = 7
print (a)
→ 7.

Assignment does not copy a value; it just attaches a name to the object that contains data. The name is reference to a thing rather than the thing itself.

Q3 Explain the rules applied in naming a Variables?

Ans: Rules applied in naming a Varibles are

1) Variable names can only contain these characters:
   - Lowercase letters (a to z)
   - Uppercase letters (A to Z)
   - Digits (0 to 9)
   - Underscore (_)

2) Name cannot begin digit. Python treats names that begin with an underscore in spcial ways. These are valid names:
   - a
   - a1
   - a_b_c__95
   - _abc
   - a_1a.

3) These names however are not Valid:

- 1
- 1a
- 1_

4) Dont use any of these keywords / reserved words as variable names:
"False class finally is return None continue for try and or global ..... break ...."

Q4. Explain Bases used in python ?

Ans:- Integers are assumed to be decimal (base 10) unless you use a prefix to specify another base.

A base is how many digits you can use until you need to "carry the one". In base 2 (binary), the only digit are 0 & 1. 0 is the same as a plain old decimal 0, and 1 is the same as a decimal 1. However in base 2 if you add a 1 to a 1, you get 10.

In python three bases besides decimals

- 0b or 0B for binary (base 2)
- 0o or 0O for octal (base 8)
- 0x or 0X for hex (base 16)

Eg: 1) plain old decimal 10. , which means 1 ten and 0 ones.

```
>>> 10
10
```

Eg2) a binary (base two), which means 1 (decimal) two and 0 ones.

```
>>> 0b10
2.
```

Eg3) Octal (base 8) for 1 (decimal) eight and 0 ones

```
>>> 0o10
8.
```

Eg 4) Hexadecimal (base 16) for 1 (decimal) 16 and 0 ones

```
>>> 0x10
16
```

The Hexadecimal uses 0 - 9 and a - f. for representing ~~16 bases~~ decimal 16.

---

05 Explain i) Escape Sequence
ii) combining with (+)
iii) Duplicate with (*)
iv) Extract a character with pair of [ ]
v) Slice with [ ]

Ans: i) Escape Sequence:

Python lets you to escape the meaning of some characters within strings to achieve effects that would otherwise be hard to express. By preceding a character with a backslash (\) you give it a special meaning. The most commonly escape sequence is \n, which means to begin a new line. with this you can create multiline strings from a one-line string

Eg: palindrome = 'A man,\n A plan, \n A canal: \n Panama'
print (palindrome)
A man,
A plan,
A canal:
Panama

you will see escape sequence \t (tab) used to align text:

Eg: print ('a \t b \t cd)
a b cd

you might also need \' or \" to specify a literal single or double quote inside a string that's quoted by same character.

Eg, testimony = "\"I did nothing!\" he said. \"Not that either! or the other thing.\""

print (testimony)

"I did nothing!" he said. "Not that either! Or the other thing."

Eg: fact = "The world's largest rubber duck was 54'2\" by 65'7\" by 105'"

print (fact)

The world's largest rubber duck was 54'2" by 65'7" by 105'

And if you need a literal backslash, just type two them.

Speech = 'Today we honor our friend, the blackslash:\\.'

print (speech)

Today we honor our friend, the blackslash:\

ii) Combine with +

You can combine literal Strings or String variables in Python by using + operator, as demonstrated here:

Eg: 'Release the kraken!' + 'At once!'
→ 'Release the kraken! At once!'

OR

"Release the kraken!" "At once!"
→ Release the kraken! At once

Eg2: a = "Duck!"
b = "a"
c = "Grey Duck!"
>>> a+b+c
Duck! DUCK! Grey Duck

iii) Duplicate with *

The operator (*) is used to duplicate a String

Eg: Start = 'Na' * 4 + 'In'
middle = 'Hey' * 3 + 'In'
end = 'Goodbye!'
print (Start + Start + middle + end)
Na Na Na Na Hey Hey Hey Goodbye.