Python Programming

Python Data Types

Beautiful is better than ugly. Explicit is better than implicit. Simple is better than complex. Complex is better than complicated. Flat is better than nested. Sparse is better than dense. Readability counts. Special cases aren't special enough to break the rules. Although practicality beats purity. Errors should never pass silently. Unless explicitly silenced. In the face of ambiguity, refuse the temptation to guess. There should be one — and preferably only one — obvious way to do it. Although that way may not be obvious at first unless you're Dutch. Now is better than never. Although never is often better than right now. If the implementation is hard to explain, it's a bad idea. If the implementation is easy to explain, it may be a good idea. Namespaces are one honking great idea — let's do more of those!
python
TM

HANDOUT #03

ISHAN VIRANTHA

B.Sc. Engineering (Hons) AM IESL

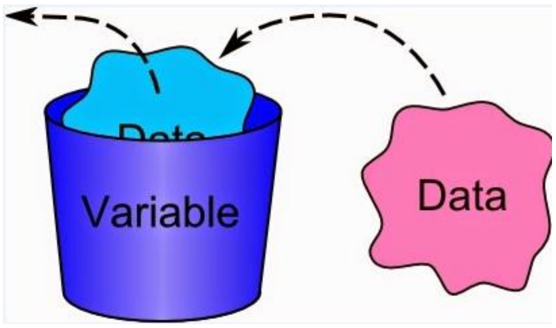
FIRST YEAR IN ENGINEERING

Tel:-071 429 39 50

3. Python Data Types

3.1 Python Variables

Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory.Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.



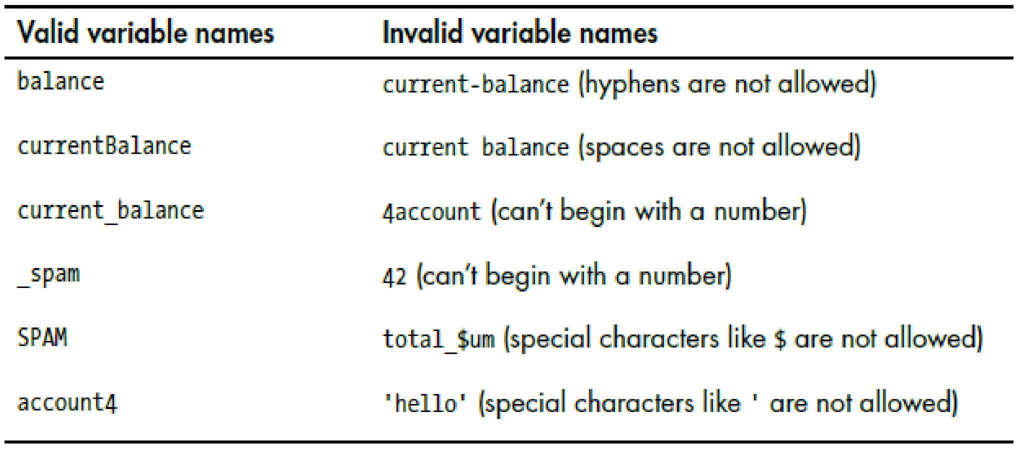
Defining python Variables

In Python When defining the variable no need to sfecifyt the data going to be store in the variable like C programing.

```
# Defining python Variables

counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string
print (counter)
print (miles)
print (name)

#####################################
# Program output

100
1000.0
John
```

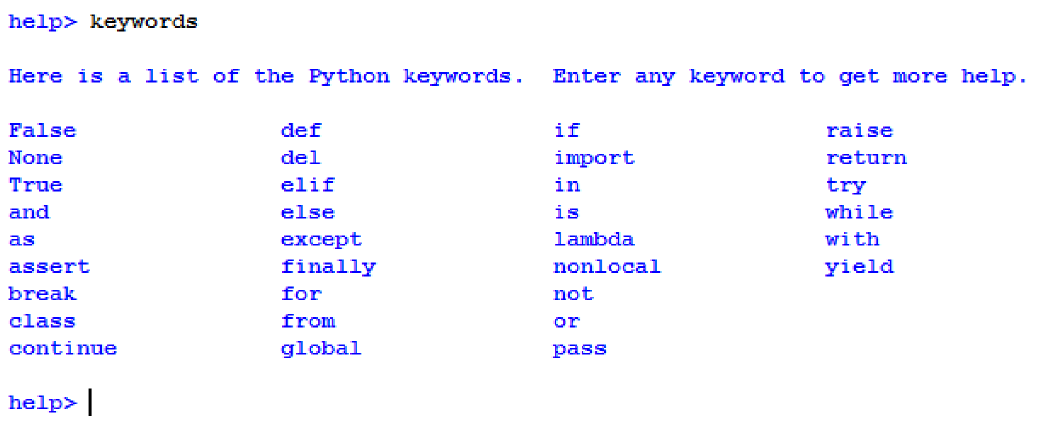
Python Variable Names

Following table has examples of legal variable names. You can name a variable anything as long as it obeys the following three rules:
1. It can be only one word.
2. It can use only letters, numbers, and the underscore (_) character.
3. It can’t begin with a number.



Valid variable names	Invalid variable names
balance	current-balance (hyphens are not allowed)
currentBalance	current balance (spaces are not allowed)
current_balance	4account (can’t begin with a number)
_spam	42 (can’t begin with a number)
SPAM	total_$um (special characters like $ are not allowed)
account4	'hello' (special characters like ' are not allowed)

List of Python key word as follows



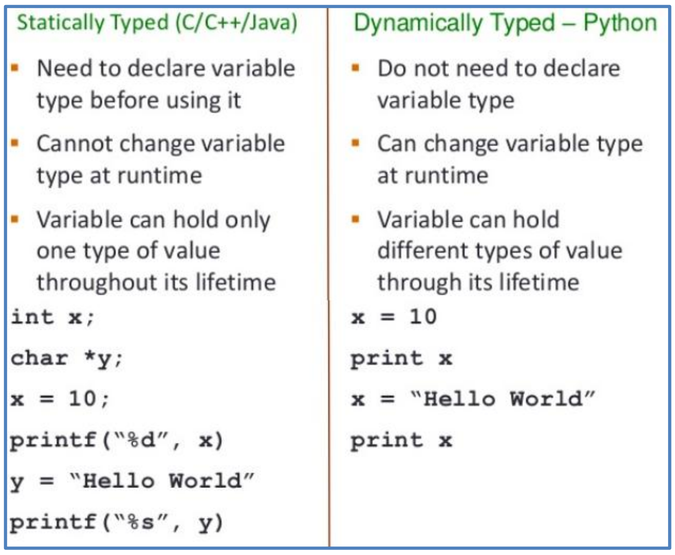
```
help> keywords

Here is a list of the Python keywords.  Enter any keyword to get more help.

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

help> |
```

Type help() in python shell and type keywords to ge the list of key word in python

Python variables are *dynamically typed*

There are two types of variables in programing, *static typed and* *dynamic typed .static typed* languages(c, c++ ,c# ,java..) are those in which *type* checking is done at compile-time, whereas *dynamic typed* languages(Python,) are those in which *type* checking is done at run-time.

Dynamically typed vs Statically typed



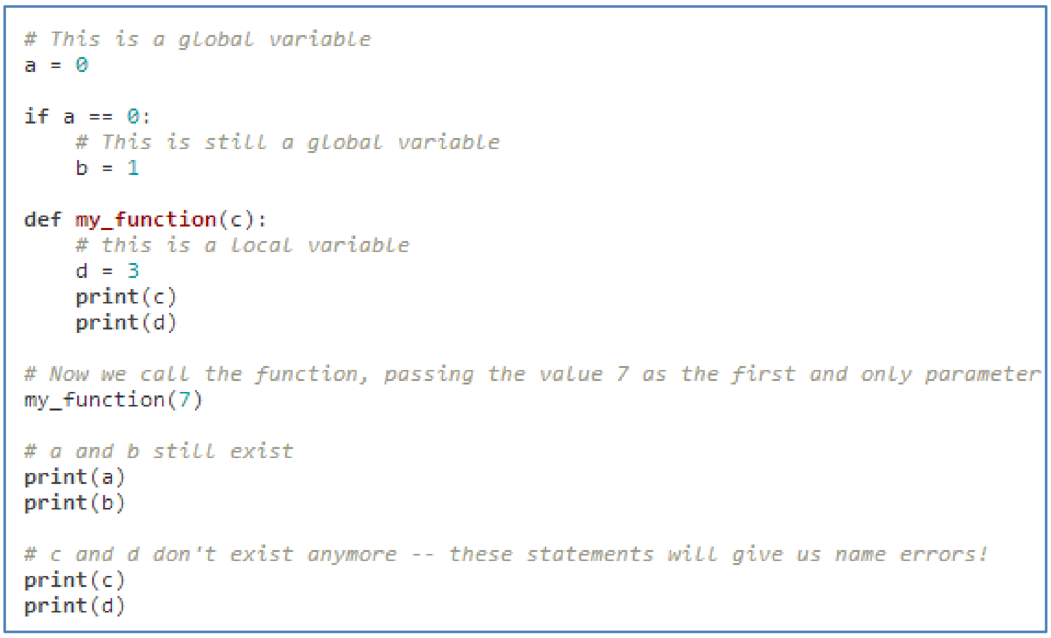
Statically Typed (C/C++/Java)	Dynamically Typed – Python
▪ Need to declare variable type before using it	▪ Do not need to declare variable type
▪ Cannot change variable type at runtime	▪ Can change variable type at runtime
▪ Variable can hold only one type of value throughout its lifetime	▪ Variable can hold different types of value through its lifetime
`int x;`	`x = 10`
`char *y;`	`print x`
`x = 10;`	`x = "Hello World"`
`printf("%d", x)`	`print x`
`y = "Hello World"`	
`printf("%s", y)`	

Scope of a Python Variable

Variables can only be seen or accessible within their scope. Two variable scopes are classified in Python:

Local variables are defined and only used by the calling function. They are not recognized by the main program or in any other calling functions.

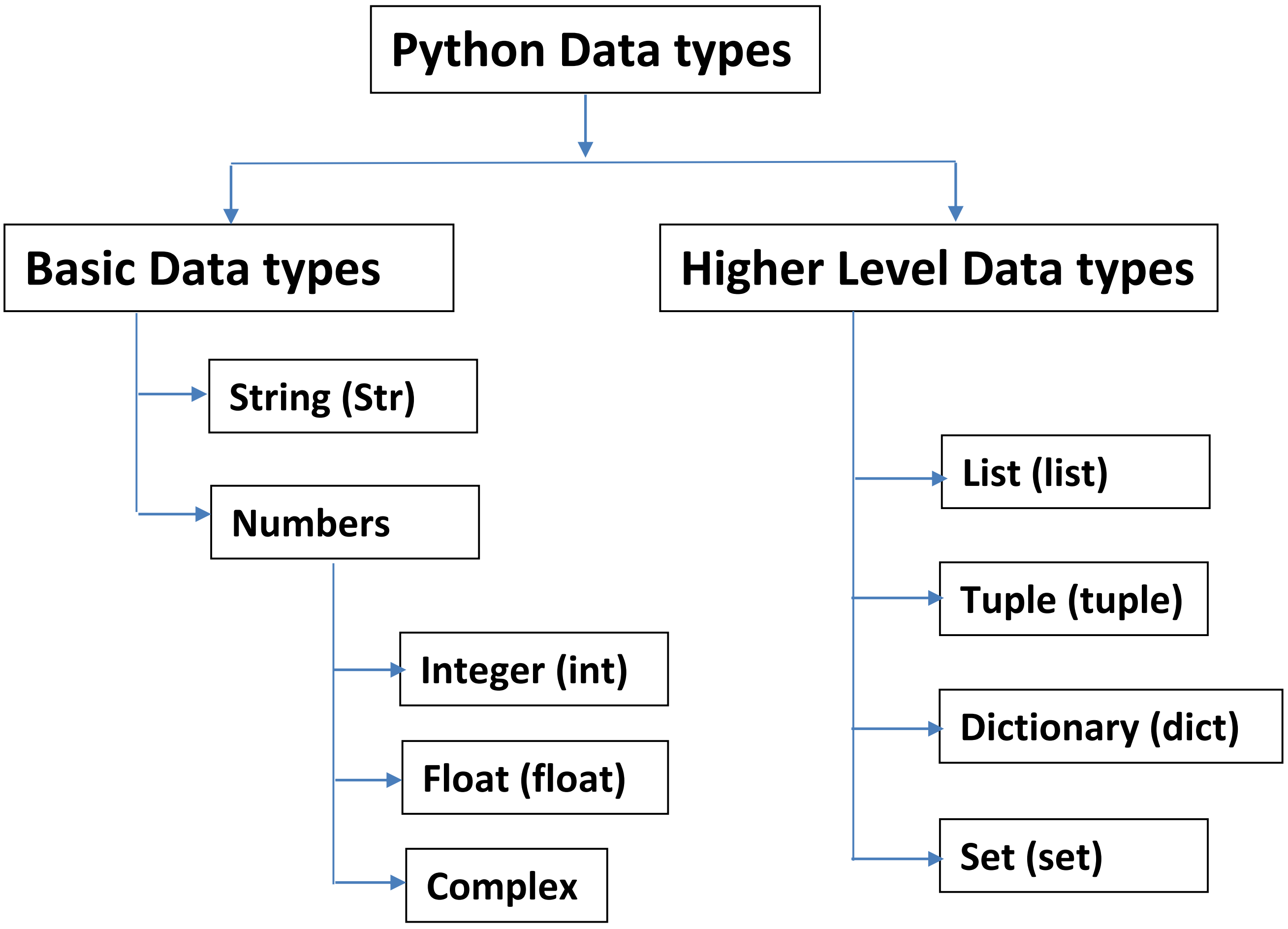
Global variables are common to most part of the code. They are recognized everywhere and can be accessed by any functions.



```python
# This is a global variable
a = 0

if a == 0:
    # This is still a global variable
    b = 1

def my_function(c):
    # this is a local variable
    d = 3
    print(c)
    print(d)

# Now we call the function, passing the value 7 as the first and only parameter
my_function(7)

# a and b still exist
print(a)
print(b)

# c and d don't exist anymore -- these statements will give us name errors!
print(c)
print(d)
```

3.2 Python Data types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as string. Python has various standard data types, can be listed as follows



String (Str)

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows either pair of single or double quotes.

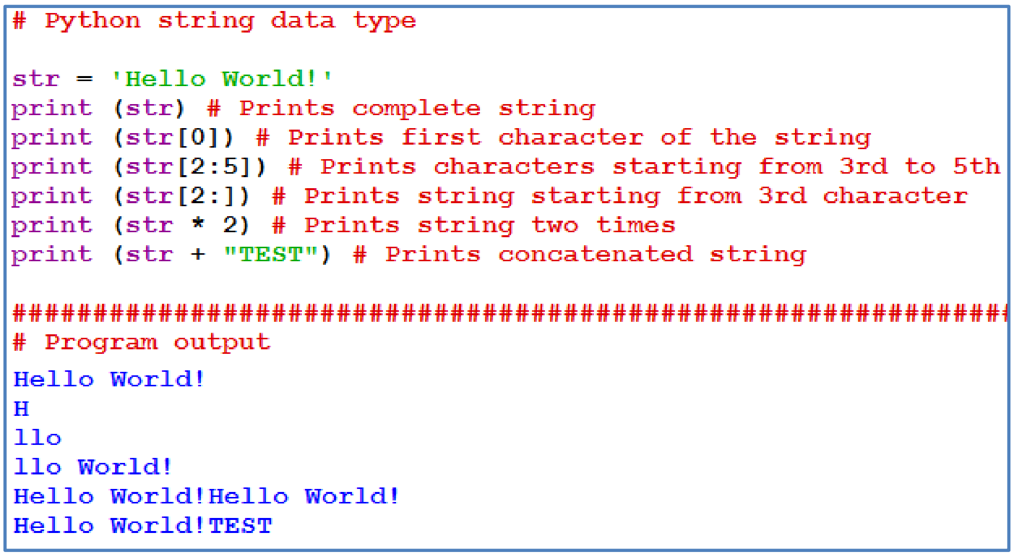
1. <u>Creating string</u>

Creating strings is as simple as assigning a value to a variable

```
# creating string variables

var1 = 'Hello World!'
var2 = "Python Programming"

print(var1)
print(var2)

#################################
# Program output

Hello World!
Python Programming
```

2. Accessing Values in Strings.

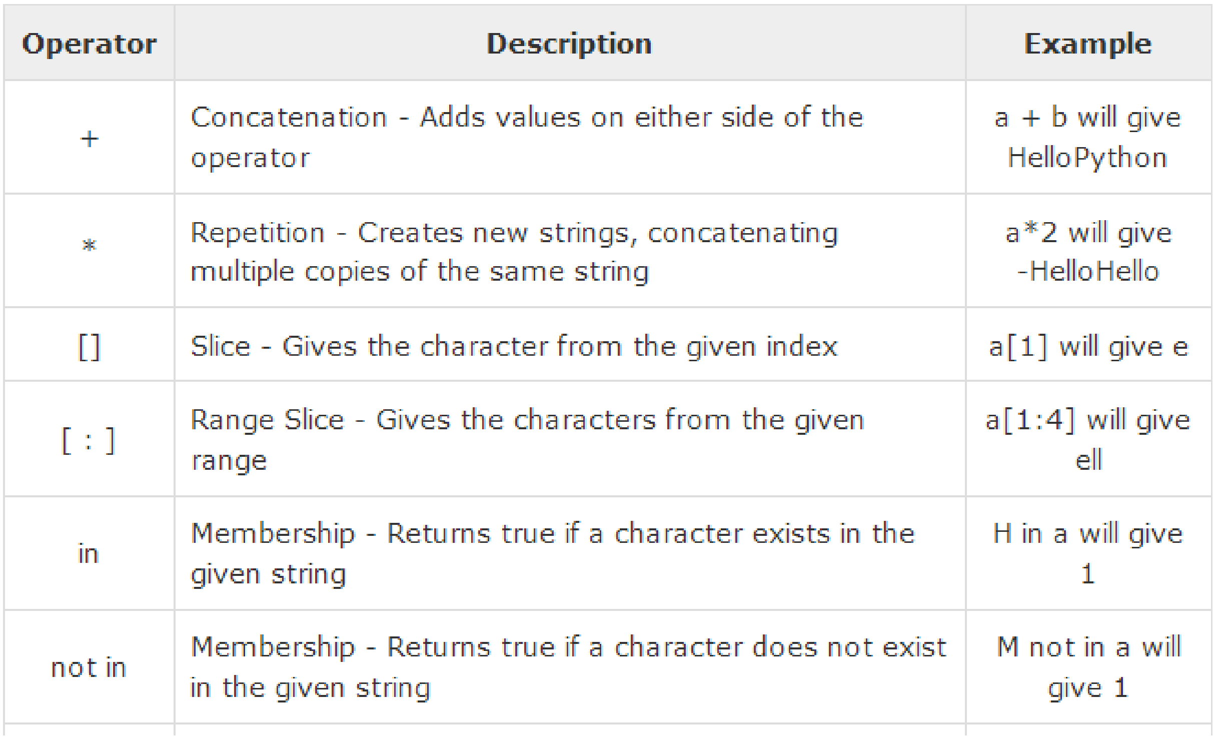
In python all string can acces through the variable nameand the part of string can acces through indexing



```
# Python string data type

str = 'Hello World!'
print (str) # Prints complete string
print (str[0]) # Prints first character of the string
print (str[2:5]) # Prints characters starting from 3rd to 5th
print (str[2:]) # Prints string starting from 3rd character
print (str * 2) # Prints string two times
print (str + "TEST") # Prints concatenated string

#############################################################
# Program output
Hello World!
H
llo
llo World!
Hello World!Hello World!
Hello World!TEST
```

3. String Special Operators

Assume string variable a holds 'Hello' and variable b holds 'Python', then

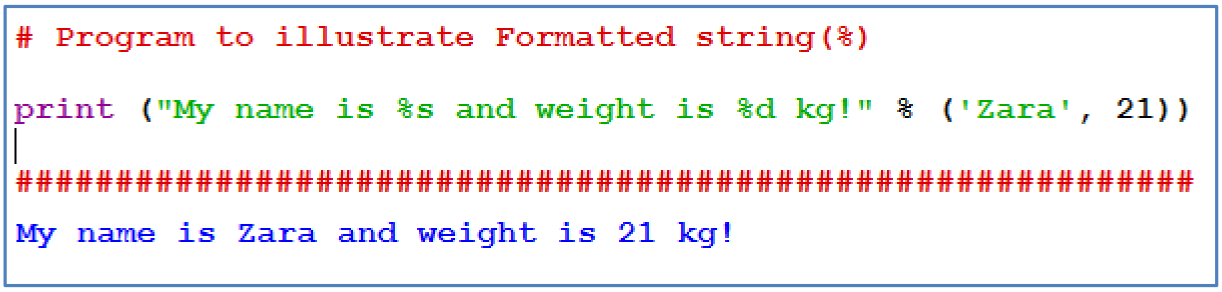


Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1

4. String formating Operator (%)

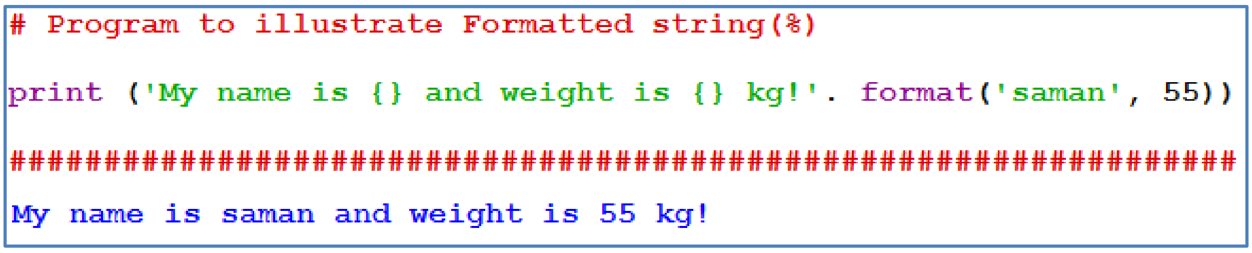
Sring format operater is used when need to insert the value of a variable, or expression into a string, For example, you can insert the value of a Decimal value into a string to display it to the user as a single string:

Python old style example



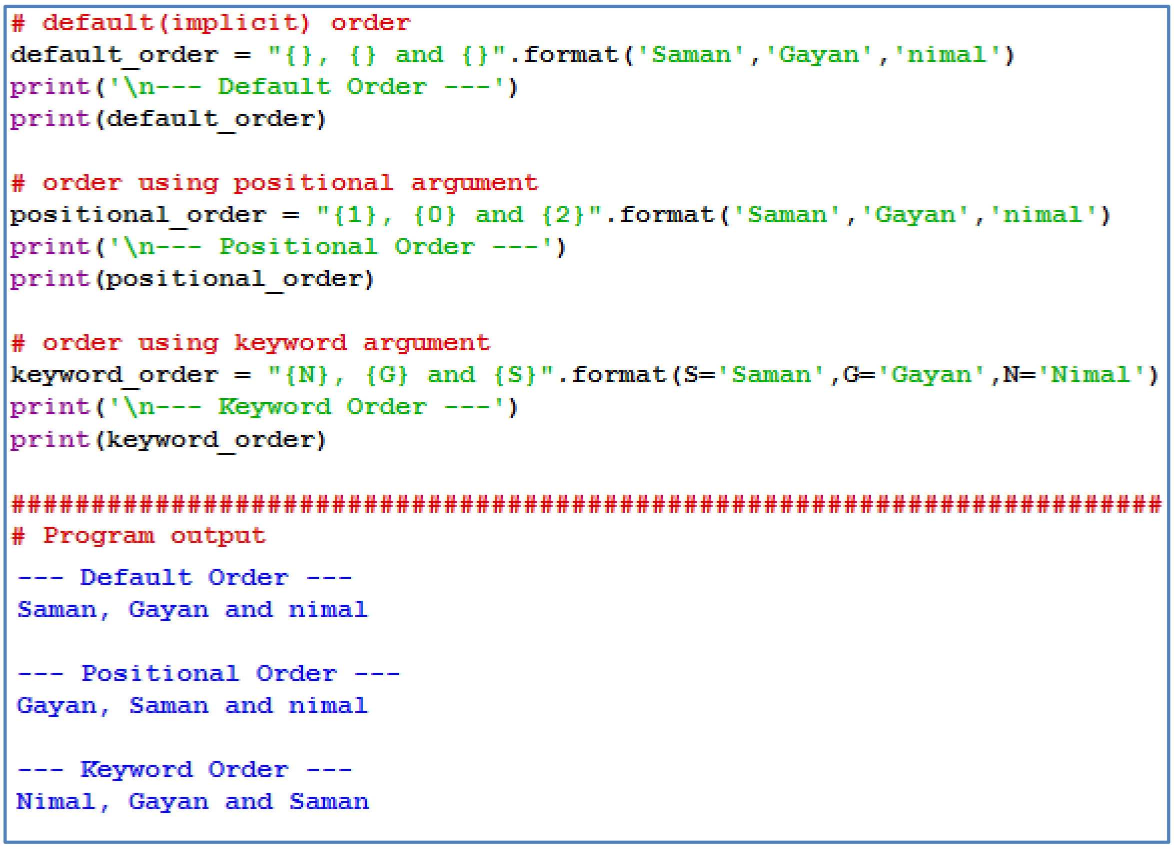
```
# Program to illustrate Formatted string(%)

print ("My name is %s and weight is %d kg!" % ('Zara', 21))

############################################################
My name is Zara and weight is 21 kg!
```

Python new style example



```
# Program to illustrate Formatted string(%)

print ('My name is {} and weight is {} kg!'. format('saman', 55))

#################################################################
My name is saman and weight is 55 kg!
```

The order of inserting variable value in a string can change using positional order or key word order. See the following example.



```
# default(implicit) order
default_order = "{}, {} and {}".format('Saman','Gayan','nimal')
print('\n--- Default Order ---')
print(default_order)

# order using positional argument
positional_order = "{1}, {0} and {2}".format('Saman','Gayan','nimal')
print('\n--- Positional Order ---')
print(positional_order)

# order using keyword argument
keyword_order = "{N}, {G} and {S}".format(S='Saman',G='Gayan',N='Nimal')
print('\n--- Keyword Order ---')
print(keyword_order)

########################################################################
# Program output
 --- Default Order ---
 Saman, Gayan and nimal

 --- Positional Order ---
 Gayan, Saman and nimal

 --- Keyword Order ---
 Nimal, Gayan and Saman
```

Here is the list of complete set of symbols which can be used along with % (**old style**)

S.No.	Format Symbol & Conversion
1	**%c** character
2	**%s** string conversion via str() prior to formatting
3	**%i** signed decimal integer
4	**%d** signed decimal integer
5	**%u** unsigned decimal integer
6	**%o** octal integer
7	**%x** hexadecimal integer (lowercase letters)
8	**%X** hexadecimal integer (UPPERcase letters)
9	**%e** exponential notation (with lowercase 'e')
10	**%E** exponential notation (with UPPERcase 'E')
11	**%f** floating point real number

Old	new	output
Basic formatting		
`'%s %s' % ('one', 'two')`	`'{} {}'.format('one', 'two')`	`one two`
`'%d %d' % (1, 2)`	`'{} {}'.format(1, 2)`	`1 2`
Padding and Aligning of strings		
`'%10s' % ('test',)`	`'{:>10}'.format('test')`	`xxxxxxtest`
`'%-10s' % ('test',)`	`'{:10}'.format('test')`	`testxxxxxx`
N/A	`'{:_<10}'.format('test')`	`test______`
N/A	`'{:^10}'.format('test')`	`xxxtestxxx`
Numbers		
`'%d' % (42)`	`'{:d}'.format(42)`	`42`
`'%f' % (3.14159265,)`	`'{:f}'.format(3.14159265)`	`3.141593`
Padding and Aligning of Numbers		
`'%4d' % (42)`	`'{:4d}'.format(42)`	`xx42`
`'%06.2f' % (3.1415926)`	`'{:06.2f}'.format(3.1415926)`	`003.14`
`'%04d' % (42,)`	`'{:04d}'.format(42)`	`0042`

5. <u>Escap character</u>

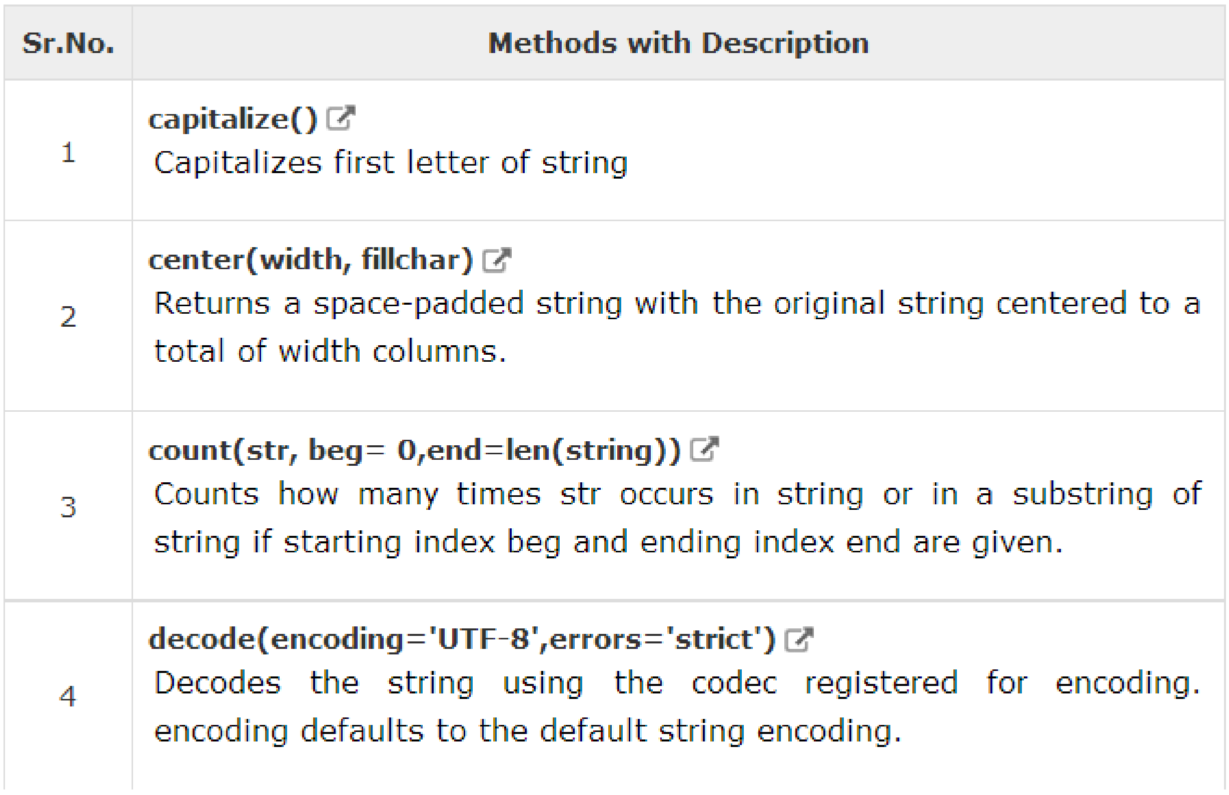
An escape character is a character which do an alternative opeartion at where it placed in a character sequence. Following table is a list of escape characters that can be represented with backslash notation.

Escape Sequence	Description	Example	Output
\\	Prints Backslash	print ("\\")	\
\'	Prints single-quote	print ("\'")	'
\"	Pirnts double quote	print ("\"")	"
\a	Bell alert sounds (eg. xterm)	print ("\a")	(bell soud)

Escape Sequence	Description	Example	Output
\b	Removes previous character	print ("ab" + "\b" + "c")	ac
\f	ASCII formfeed (FF)	print ("hello\fworld")	hello world
\n	ASCII linefeed (LF)	print ("hello\nworld")	hello world
\t	ASCII horizontal tab (TAB). Prints TAB	print ("\t* hello")	* hello

6. Built-in String Methods

Python includes the following built-in methods (fucntions) to manipulate strings.



Sr.No.	Methods with Description
1	**capitalize()** Capitalizes first letter of string
2	**center(width, fillchar)** Returns a space-padded string with the original string centered to a total of width columns.
3	**count(str, beg= 0,end=len(string))** Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
4	**decode(encoding='UTF-8',errors='strict')** Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.

Numbers

Number data types store numeric values. Number objects are created when you assign a value to them.Python supports three different numerical types –

□ int (signed integers)
□ float (floating point real values)
□ complex (complex numbers)

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3+e18	.876j
-0490	-90.	-.6545+0J
-0x260	-32.54e100	3e+26J
0x69	70.2-E12	4.53e-7j

A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are real numbers and j is the imaginary unit.

List (list)

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One of the differences between them is that all the items belonging to a list can be of different data type.

7. How to create a list?

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]
```

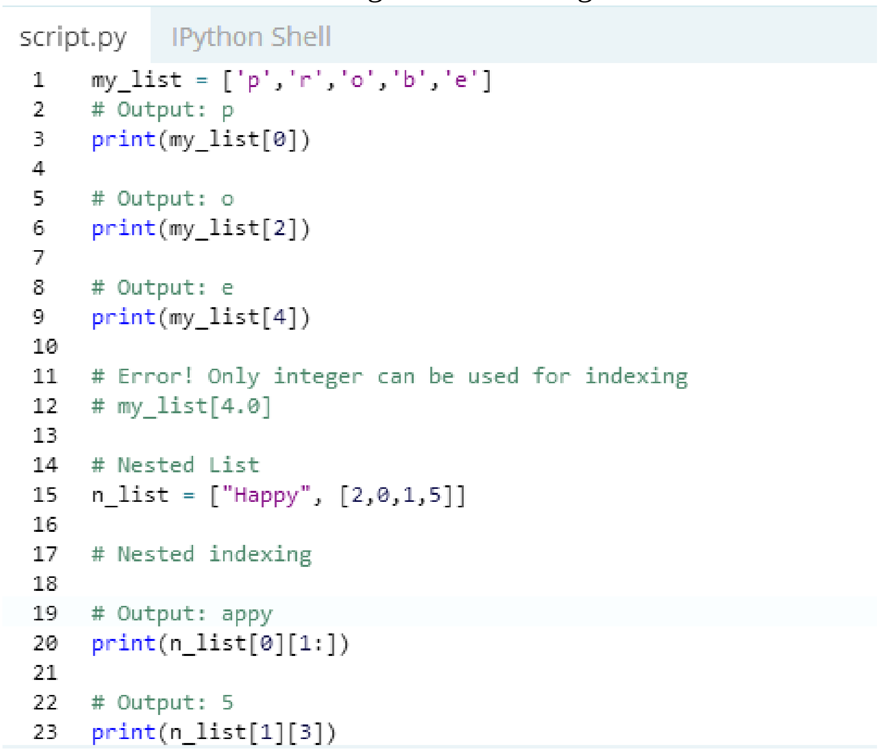
Also, a list can even have another list as an item. This is called nested list.

```
# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

8. **How to access elements from a list?**

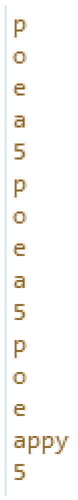
We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4. The index must be an integer.

Nested list are accessed using nested indexing.



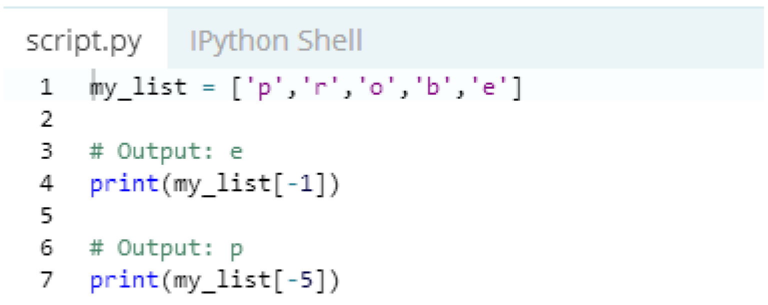
```python
my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Error! Only integer can be used for indexing
# my_list[4.0]

# Nested List
n_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: appy
print(n_list[0][1:])

# Output: 5
print(n_list[1][3])
```

Output will be as follows



```
p
o
e
a
5
p
o
e
a
5
p
o
e
appy
5
```

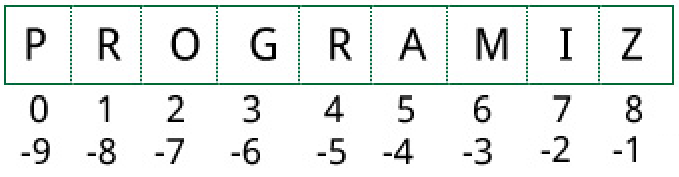
Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

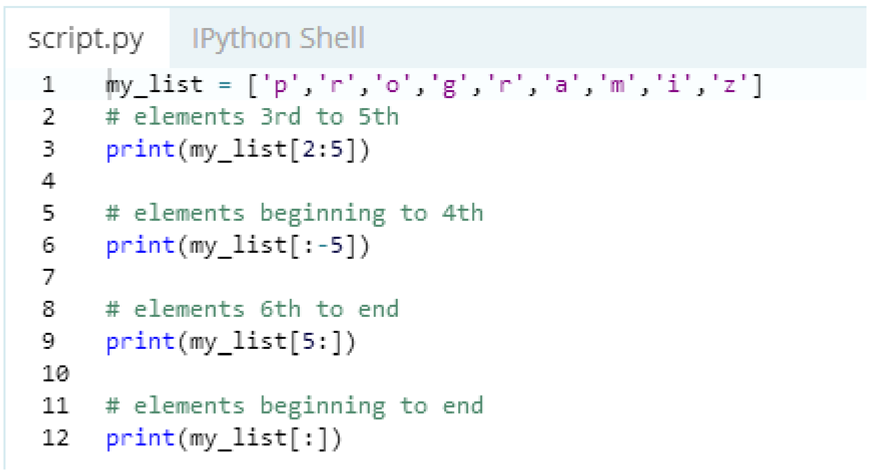


```python
my_list = ['p','r','o','b','e']

# Output: e
print(my_list[-1])

# Output: p
print(my_list[-5])
```

9. How to slice lists in Python?

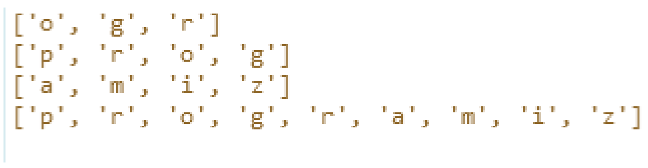
We can access a range of items in a list by using the slicing operator (colon).





```python
my_list = ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

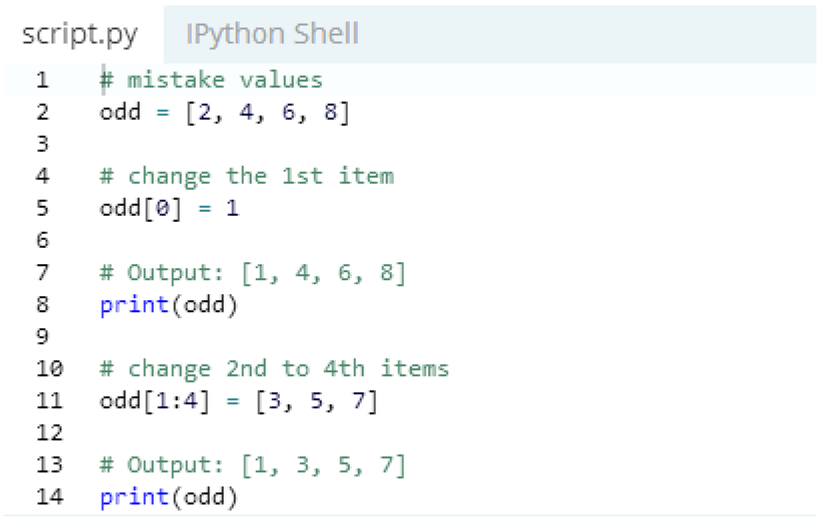
Output will be as follows



```
['o', 'g', 'r']
['p', 'r', 'o', 'g']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

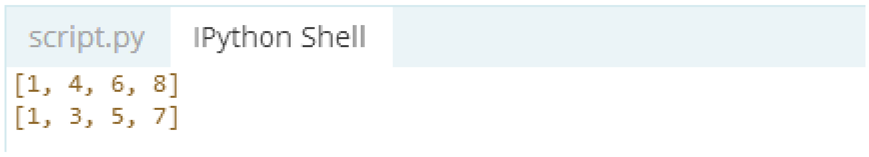
10. How to change or add elements to a list?

List are mutable, meaning, their elements can be changed unlike string or tuple. We can use assignment operator (=) to change an item or a range of items.



```
# mistake values
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

# Output: [1, 4, 6, 8]
print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]
print(odd)
```

Output will be as follows



```
[1, 4, 6, 8]
[1, 3, 5, 7]
```

We can add one item to a list using append() method or add several items using extend() method.

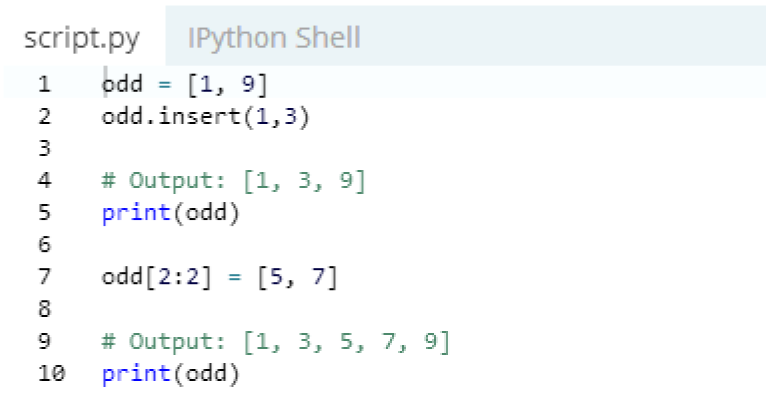
```
odd = [1, 3, 5]

odd.append(7)

# Output: [1, 3, 5, 7]
print(odd)

odd.extend([9, 11, 13])

# Output: [1, 3, 5, 7, 9, 11, 13]
print(odd)
```

We can also use + operator to combine two lists. This is also called concatenation.

The * operator repeats a list for the given number of times.

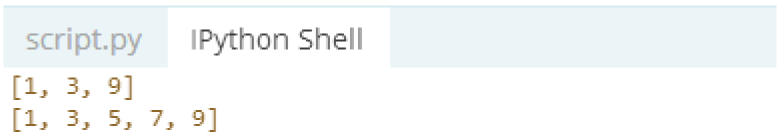
```
odd = [1, 3, 5]

# Output: [1, 3, 5, 9, 7, 5]
print(odd + [9, 7, 5])

#Output: ["re", "re", "re"]
print(["re"] * 3)
```

Furthermore, we can insert one item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.



```python
odd = [1, 9]
odd.insert(1,3)

# Output: [1, 3, 9]
print(odd)

odd[2:2] = [5, 7]

# Output: [1, 3, 5, 7, 9]
print(odd)
```

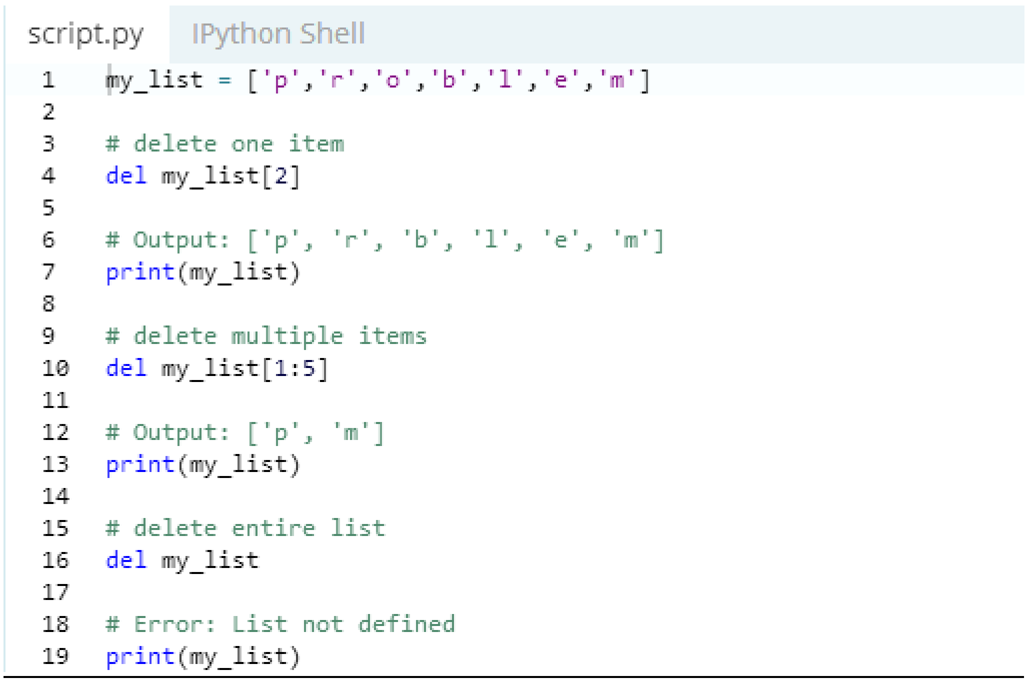
Output will be as follows



```
[1, 3, 9]
[1, 3, 5, 7, 9]
```

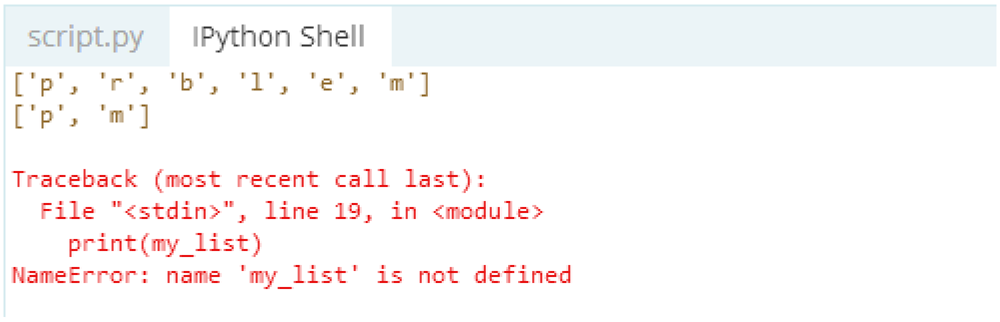
11. How to delete or remove elements from a list?

We can delete one or more items from a list using the keyword **del**. It can even delete the list entirely.



```python
my_list = ['p','r','o','b','l','e','m']

# delete one item
del my_list[2]

# Output: ['p', 'r', 'b', 'l', 'e', 'm']
print(my_list)

# delete multiple items
del my_list[1:5]

# Output: ['p', 'm']
print(my_list)

# delete entire list
del my_list

# Error: List not defined
print(my_list)
```

Output will be as follows



```
['p', 'r', 'b', 'l', 'e', 'm']
['p', 'm']

Traceback (most recent call last):
  File "<stdin>", line 19, in <module>
    print(my_list)
NameError: name 'my_list' is not defined
```

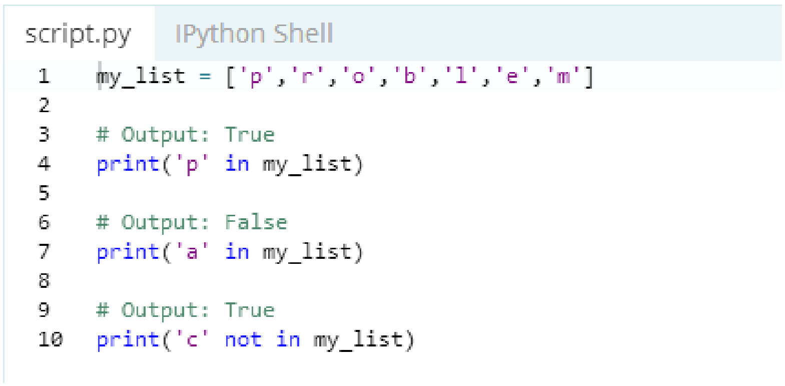
We can use `remove()` method to remove the given item or `pop()` method to remove an item at the given index.

The `pop()` method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).

We can also use the `clear()` method to empty a list.

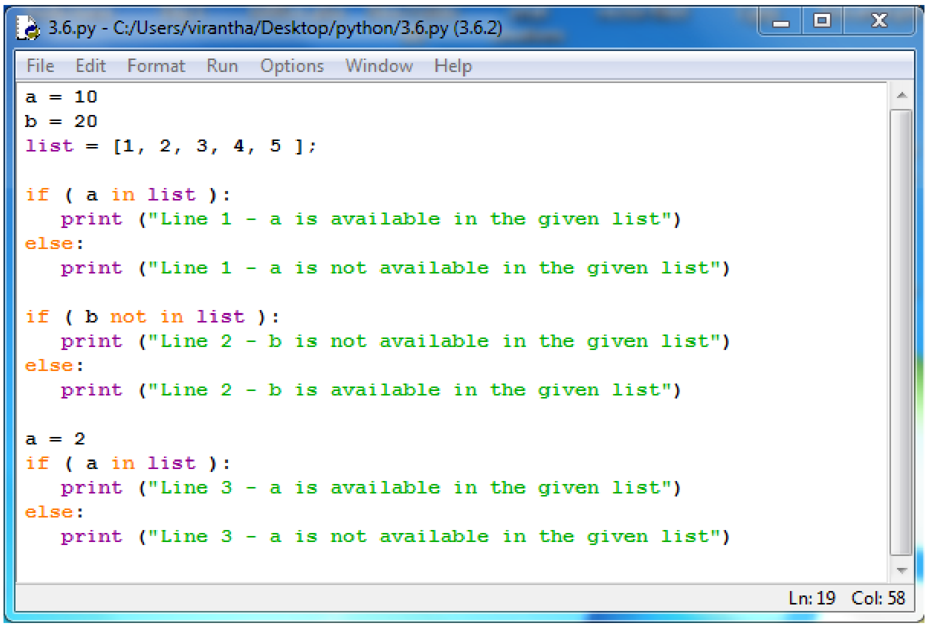
12. <u>Python List Membership Test</u>

We can test if an item exists in a list or not, using the keyword in.



```
my_list = ['p','r','o','b','l','e','m']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)
```

Example program 3.6


3.6.py - C:/Users/virantha/Desktop/python/3.6.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 19 Col: 58

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print ("Line 1 - a is available in the given list")
else:
   print ("Line 1 - a is not available in the given list")

if ( b not in list ):
   print ("Line 2 - b is not available in the given list")
else:
   print ("Line 2 - b is available in the given list")

a = 2
if ( a in list ):
   print ("Line 3 - a is available in the given list")
else:
   print ("Line 3 - a is not available in the given list")
```

Example program 3.6 output

13. Iterating Through a List

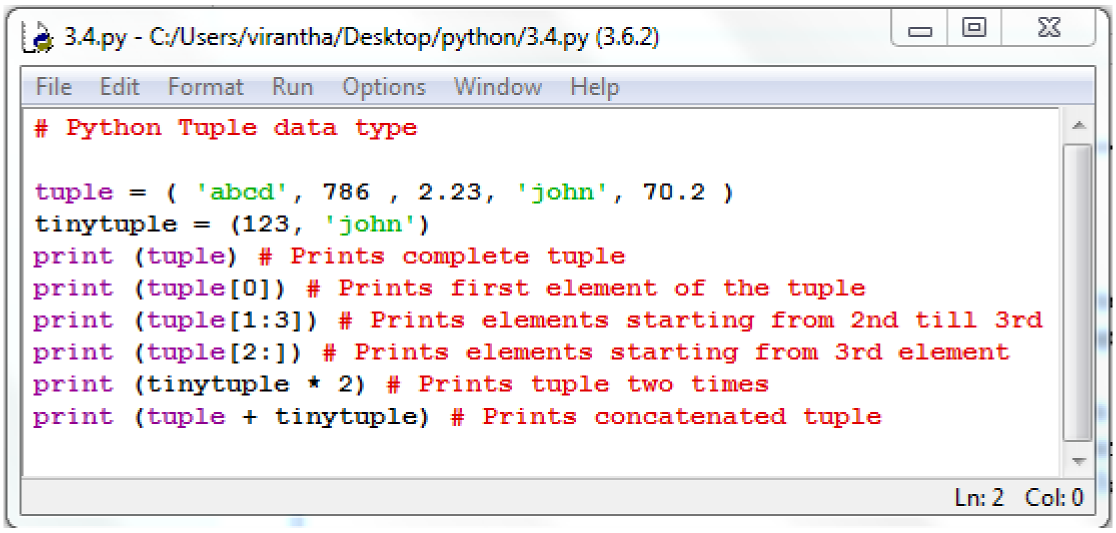
NOTE1	NOTE2

IN C:
 	
`for (i = 0; i < n; i++)`
 	
In python3:
 	
`for i in range(n):`
	In C:
	
	`for(int i=0; i<9; i+=2)`
	`{`
	`    dosomething(i);`
	`}`
	
	In python3:
	
	`for i in range(0, 9, 2):`
	`    dosomething(i)`

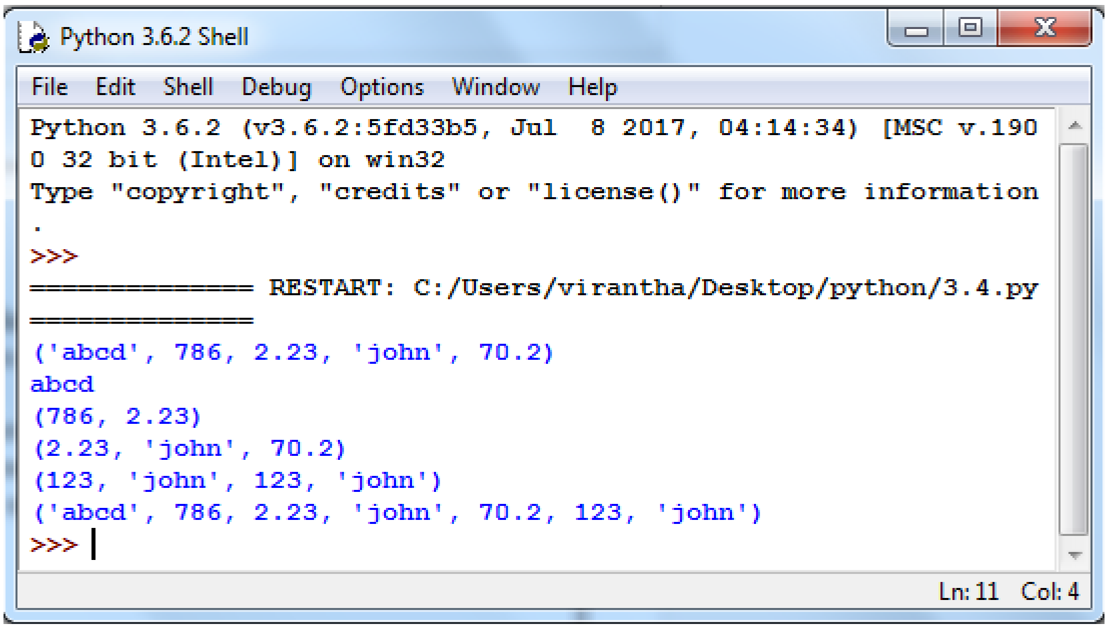
Tuple (tuple)

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 parenthesis.

The main difference between lists and tuples is- Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated. Tuples can be thought of as **read-only** lists. For example-

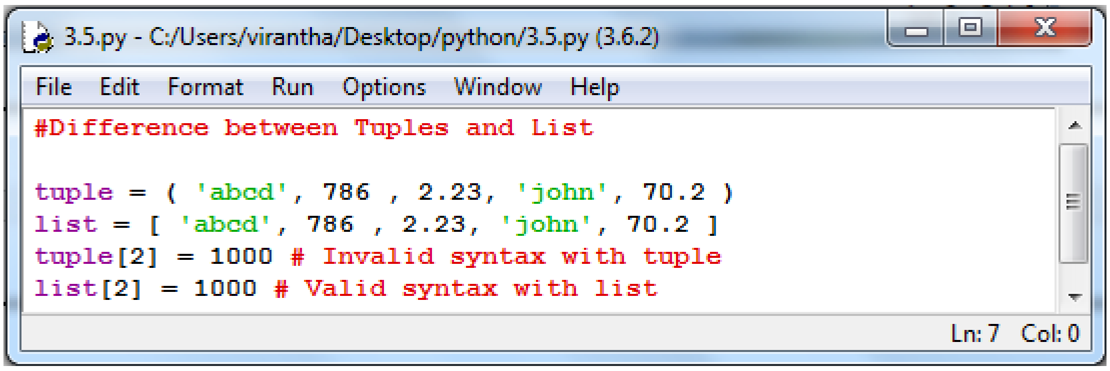

3.4.py - C:/Users/virantha/Desktop/python/3.4.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 2 Col: 0

```
# Python Tuple data type

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tinytuple = (123, 'john')
print (tuple) # Prints complete tuple
print (tuple[0]) # Prints first element of the tuple
print (tuple[1:3]) # Prints elements starting from 2nd till 3rd
print (tuple[2:]) # Prints elements starting from 3rd element
print (tinytuple * 2) # Prints tuple two times
print (tuple + tinytuple) # Prints concatenated tuple
```



```
Python 3.6.2 (v3.6.2:5fd33b5, Jul  8 2017, 04:14:34) [MSC v.190
0 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information
.
>>>
============== RESTART: C:/Users/virantha/Desktop/python/3.4.py
==============
('abcd', 786, 2.23, 'john', 70.2)
abcd
(786, 2.23)
(2.23, 'john', 70.2)
(123, 'john', 123, 'john')
('abcd', 786, 2.23, 'john', 70.2, 123, 'john')
>>> 
```

Difference between Tuples and List


3.5.py - C:/Users/virantha/Desktop/python/3.5.py (3.6.2)
File Edit Format Run Options Window Help
Ln: 7 Col: 0

```
#Difference between Tuples and List

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
tuple[2] = 1000 # Invalid syntax with tuple
list[2] = 1000 # Valid syntax with list
```

Python Dictionary Data type

Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair. While values can be of any data type and can repeat, keys must be of immutable (unable to change) type (string, number or tuple with immutable elements) and must be unique.

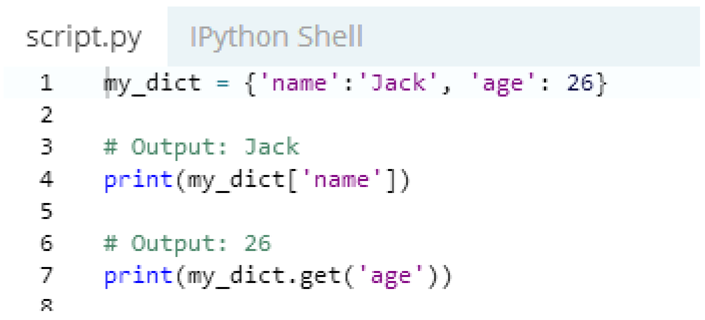
➢ **Creating a dictionary is as simple as placing items inside curly braces {} separated by comma**.

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

As you can see above, we can also create a dictionary using the built-in function `dict()`.

➢ How to access elements from a dictionary?

Dictionary uses keys accsess the data . Key can be used either inside square brackets or with the get() method.



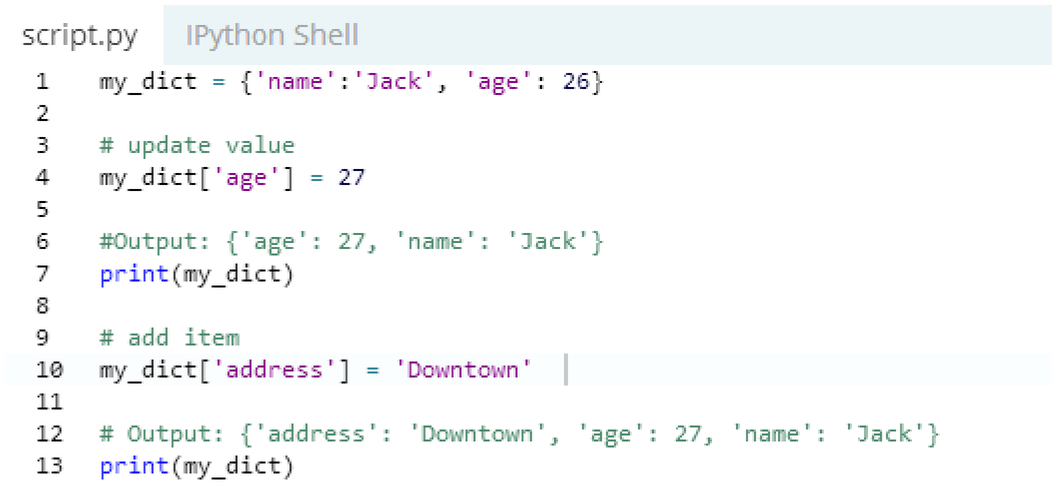
```python
my_dict = {'name':'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))
```

When you run the program, the output will be:

```
Jack
26
```

➢ How to change or add elements in a dictionary?

We can add new items or change the value of existing items using assignment operator.If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.



```python
my_dict = {'name':'Jack', 'age': 26}

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)
```

When you run the program, the output will be:

```
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

FIRST YEAR IN ENGINEERING

Tel:- 071 429 39 50