Python_Variable_Practice

Important Function in Python

- help()
- type()
- print()
- dir()

In [ ]:
```python
# print() -printing data
# type() -  verifying data type
# dir() - getting inforation about supporting functions or methods or classes....
# help() - complete note about object
```

In [0]:
```python
# dir
list = [1,2,3]
dir(list)
```

```
Out[23]: ['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'append',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

In [5]:
```python
### Print
print('this is sample print function')
```

```
this is sample print function
```

In [7]:
```python
#Help
List = [1,2,3]
help(List)
```

```
Help on list object:

class list(object)
 |  list(iterable=(), /)
 |
 |  Built-in mutable sequence.
 |
 |  If no argument is given, the constructor creates a new empty list.
 |  The argument must be an iterable if specified.
 |
 |  Methods defined here:
 |
 |  __add__(self, value, /)
 |      Return self+value.
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __delitem__(self, key, /)
 |      Delete self[key].
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(...)
 |      x.__getitem__(y) <==> x[y]
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __iadd__(self, value, /)
 |      Implement self+=value.
 |
 |  __imul__(self, value, /)
 |      Implement self*=value.
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __reversed__(self, /)
 |      Return a reverse iterator over the list.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  __setitem__(self, key, value, /)
 |      Set self[key] to value.
 |
 |  __sizeof__(self, /)
 |      Return the size of the list in memory, in bytes.
 |
 |  append(self, object, /)
 |      Append object to the end of the list.
 |
 |  clear(self, /)
 |      Remove all items from list.
 |
 |  copy(self, /)
 |      Return a shallow copy of the list.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  extend(self, iterable, /)
 |      Extend list by appending elements from the iterable.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  insert(self, index, object, /)
 |      Insert object before index.
 |
 |  pop(self, index=-1, /)
 |      Remove and return item at index (default last).
 |
 |      Raises IndexError if list is empty or index is out of range.
 |
 |  remove(self, value, /)
 |      Remove first occurrence of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  reverse(self, /)
 |      Reverse *IN PLACE*.
 |
 |  sort(self, /, *, key=None, reverse=False)
 |      Sort the list in ascending order and return None.
 |
 |      The sort is in-place (i.e. the list itself is modified) and stable (i.e. the
 |      order of two equal elements is maintained).
 |
 |      If a key function is given, apply it once to each list item and sort them,
 |      ascending or descending, according to their function values.
 |
 |      The reverse flag can be set to sort in descending order.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  __hash__ = None
```

In [9]:
```python
#Type
List = [1,2,3]
type(List) # It returns us the data type
```

Out[9]:
```
list
```

what is Python Identifier?

- Python Identifier is the name we give to identify a variable, list, tuple, sets, dictionary, function, class, module or other object. That means whenever we want to give an entity a name, that's called identifier.
- Sometimes variable and identifier are often misunderstood as same but they are not. Well for clarity, let's see what is a variable?

What is a Variable in Python?

- A variable, as the name indicates is something whose value is changeable over time. In fact a variable is a memory location where a value can be stored. * Later we can retrieve the value to use. But for doing it we need to give a nickname to that memory location
- so that we can refer to it. That's identifier, the nickname.

In [10]:
```python
# upper case variable names prefarable for Data Engineer -
PATH ="/location/temp/customer"
USERNAME="admin"
PASSWORD ="admin"
PORT_NO= 23006
```

Memory allocation for integer variables

In [17]:
```python
a=100
b=555
print('a variable id is : ',id(a))
print('b variable id is : ',id(b))
```

```
a variable id is :  2165667550672
b variable id is :  2165778594160
```

In [18]:
```python
id(a)
```

Out[18]:
```
2165667550672
```

In [19]:
```python
# here you can find same object id. Bcz python memory utilization it will store all integers in memory.
# if you are creating different variable with same value. it will use existing storage location and value to avoid more memory utilization.
# this numbers will be used for range [-5,256]
c=100
d=555
print('c variable id is : ',id(c))
print('d variable id is : ',id(d))
```

```
c variable id is :  2165667550672
d variable id is :  2165778595536
```

In [22]:
```python
# inside python list 55 value also having same id.
lst =[11,22,55]
print(id(lst[2]))
```

```
2165667549232
```

In [23]:
```python
# above 256 its giving different id. So if you have any integers between range [-5,256] then it will reuse the memory.
a=300
b=300
print(id(a))
print(id(b))
```

```
2165778595600
2165778594896
```

- local variables we can create starting with v_* name
- Global variables we can create starting with gv_* name
- Global Variables example:
-
  - any source system and target paths(file locations) which is using entire project
-
  - common parameters which is unsing entire project like username and keys...
-
  - if any variable which is using in other notebooks we can create as gv_* variables.

In [0]:
```python
var="this is variable"
print('this is sample print string printing on screen using print() function ',var)
```

```
this is sample print string printing on screen using print() function  this is variable
```

In [0]:
```python
num1=55
num2='11'
print(str(num1)+num2)
```

```
5511
```

In [0]:
```python
name="ravi"
print('My Name is : ',name)
print('Type of Name variable is : ',type(name))
```

```
My Name is :  ravi
Type of Name variable is :  <class 'str'>
```

In [0]:
```python
age=33   # int variable
print(age)
name='ravi' # String variable
print(name)
# Get the variable type using type() function
print('age Variable Type is :',type(age))
print('name Variable Type is :',type(name))
```

Creating Variable and Assigning a Values...

In [27]:
```python
a=b=c=d=e=55
a='practice'
print('a value is : ',a)
print('b value is : ',b)
print('c value is : ',c)
print('d value is : ',d)
print('e value is : ',e)
```

```
a value is :  practice
b value is :  55
c value is :  55
d value is :  55
e value is :  55
```

In [29]:
```python
print('this is practice variable a Value : '+a)
# + for addition or concatication
```

```
this is practice variable a Value : practice
```

In [31]:
```python
print(type(a))
```

```
<class 'str'>
```

In [32]:
```python
NAME='RAVI','RAM'
print(NAME)
```

```
('RAVI', 'RAM')
```

In [33]:
```python
ipaddress_v='193.12.12.4'
print(ipaddress_v)
```

```
193.12.12.4
```

In [34]:
```python
id1=66
id2=77
print(id1+id2)
```

```
143
```

Strings

- Besides numbers, Python can also manipulate strings, which can be expressed in several ways. They can be enclosed in single quotes ('...') or double quotes ("...") with the same result 2. \ can be used to escape quotes:

In [36]:
```python
print('spam eggs')  # single quotes
print("doesn't")
print('"Yes," they said.')
```

```
spam eggs
doesn't
"Yes," they said.
```

Numbers

- The interpreter acts as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators +, -, * and / work just like in most other languages (for example, Pascal or C); parentheses (()) can be used for grouping. For example:

In [37]:
```python
print(6//2)
```

```
3
```

In [38]:
```python
print(2 + 2)
print(50 - 5*6)
print((50 - 5*6) / 4)
print(8 / 5)  # division always returns a floating point number
```

```
4
20
5.0
1.6
```

- Division (/) always returns a float. To do floor division and get an integer result (discarding any fractional result) you can use the `//` operator; to calculate the remainder you can use %:

In [39]:
```python
print(17 / 3)  # classic division returns a float
print(17 // 3)  # floor division discards the fractional part
print(17 % 3) # the % operator returns the remainder of the division
print(5 * 3 + 2)  # result * divisor + remainder
```

```
5.666666666666667
5
2
17
```

- it is possible to use the ** operator to calculate powers

In [40]:
```python
print (5 ** 2) # 5 Squared
print (2 ** 7) # 2 to the power of 7
```

```
25
128
```

- The equal sign (=) is used to assign a value to a variable.
- `+` for Concatenating two strings for string data type variables

In [44]:
```python
a='G '
b='Govardhan'
c= a + b # + working as Concatenation operator
print(c)
```

```
G Govardhan
```

In [45]:
```python
a = 10
b = 111
c = a+b  # + working as adition operator
print(c)
```

```
121
```

In [47]:
```python
num=10
name='Krishna'
myfloat = 5.90
print(num)
print(name)
print(myfloat)
print("num variable Data type is : ",type(num))
print('name variable Data type is : ' , type(name))
print('myfloat variable data type is : ',type(myfloat))
```

```
10
Krishna
5.9
num variable Data type is :  <class 'int'>
name variable Data type is :  <class 'str'>
myfloat variable data type is :  <class 'float'>
```

Multiple Assignment

- Python allows you to assign a single value to several variables simultaneously

In [48]:
```python
name=12
print("this is sample variable value : ",name)
```

```
this is sample variable value :  12
```

In [49]:
```python
a = b = c =55
print('a variable Value is : ',a)
print('b variable Value is : ',b)
print('c variable Value is : ',c)
```

```
a variable Value is :  55
b variable Value is :  55
c variable Value is :  55
```

In [50]:
```python
a,b=55,66
```

In [51]:
```python
V1='55'
print(type(V1))
V2='TEST'
print(type(V2))
```

```
<class 'str'>
<class 'str'>
```

In [52]:
```python
a=55
a=66
```

In [53]:
```python
num1=5
num2=9
print('The average of the numbers you entered is', (num1+num2)/2)
```

```
The average of the numbers you entered is 7.0
```

Adding two int variables.

In [54]:
```python
id=128

print('This is my TEXT Variable : ',type(id) )
```

```
This is my TEXT Variable :  <class 'int'>
```

In [55]:
```python
""" Declare a variable and initialize it"""
A = '5'
a = '10'
#printing Variable
print('Variable value is :',A)
print('Variable value is :',a)
#verifying Data Type  using TYPE()
print('Variable Type A Is : ',type(A))
print('Variable Type a Is : ',type(a))
# Adding two variables
print('Sum of A plus a : ',int(A)+int(a))
```

```
Variable value is : 5
Variable value is : 10
Variable Type A Is :  <class 'str'>
Variable Type a Is :  <class 'str'>
Sum of A plus a :  15
```

Adding String & Interger Variables

- Data Type Conversions using
- int
- str

In [56]:
```python
# Declare a variable and initialize it
A = "55"
B = 66
print(A+str(B))
```

```
5566
```

In [57]:
```python
x = int(1)   # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
print(x,y,z)
```

```
1 2 3
```

In [58]:
```python
x = float(1)     # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")   # z will be 3.0
w = float("4.2") # w will be 4.2
print(x,y,z,w)
```

```
1.0 2.8 3.0 4.2
```

In [59]:
```python
x = str("s1") # x will be 's1'
y = str(2)    # y will be '2'
z = str(3.0)  # z will be '3.0'
x,y,z
```

Out[59]:
```
('s1', '2', '3.0')
```

In [60]:
```python
# Declare a variable and initialize it
A = "55"
B = 66
C = int(A)+B
C
```

Out[60]:
```
121
```

In [61]:
```python
# Declare a variable and initialize it
A = "55"
B = 66
C = int(A)+B
#printing Variable
print('Variable value is A :',A)
print('Variable value is B :',B)
#verifying Data Type  using TYPE()
print('Variable Type A Is : ',type(A))
print('Variable Type B Is : ',type(B))
# Adding two variables
print('Sum of A plus B : ',A+str(B))
print('Type of A + STR(B) : ',type(A+str(B)))
print('Sum of A Plus B : ',C)
```

```
Variable value is A : 55
Variable value is B : 66
Variable Type A Is :  <class 'str'>
Variable Type B Is :  <class 'int'>
Sum of A plus B :  5566
Type of A + STR(B) :  <class 'str'>
Sum of A Plus B :  121
```

Delete a variable using DEL

In [63]:
```python
A=55
print(A)
del A
print(A) # Variable A is deleted so we are getting as name 'A' is not defined
```

```
55
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [63], in <cell line: 4>()
      2 print(A)
      3 del A
----> 4 print(A)

NameError: name 'A' is not defined
```

F-strings

- F-Strings provide a way to embed expressions inside string literals, using a minimal syntax. It should be noted that an f-string is really an expression evaluated at run time, not a constant value. In Python source code, an f-string is a literal string, prefixed with 'f', which contains expressions inside braces. The expressions are replaced with their values.

In [64]:
```python
table_name='customer'
total_rows=534
print("Total Number of records processed {} for {} ".format(total_rows,table_name))
```

```
Total Number of records processed 534 for customer
```

In [65]:
```python
c_count= 40
target_count=30
reject=10
fstring = f'source count is : {c_count} target count is : {target_count} rejected count is : {reject}'
print(fstring)
```

```
source count is : 40 target count is : 30 rejected count is : 10
```

In [66]:
```python
c_count= 40
target_count=30
reject=10
print('customer data processed',c_count)

print('customer data process completed and source count :{c} target count : {b} and rejected count :{a}  '.format(a=c_count,b=target_count,c=reject))
# position based
# Index based  - indexes start with zero (0)
# refrenced
```

```
customer data processed 40
customer data process completed and source count :10 target count : 30 and rejected count :40
```

In [67]:
```python
name="Suresh"
age=35
string=f'this is : {name} And My age is : {age}'
print(string)
```

```
this is : Suresh And My age is : 35
```

In [68]:
```python
name='Sai Ram'
age = 28
f_string = f'My Name is : {name} and My Age is : {age}'
print(f_string)
```

```
My Name is : Sai Ram and My Age is : 28
```

In [69]:
```python
a=55
b='ravi'
print(f'a value is : {a} , b variable value is : {b}')
fstring=f'Hi.. sample F String text A value: {a} And B Value is : {b}'
print(fstring)
```

```
a value is : 55 , b variable value is : ravi
Hi.. sample F String text A value: 55 And B Value is : ravi
```

In [70]:
```python
x=55
y=66
str = f"X variable value is: {x} test for f-strings Y variable value is : {y}"
str
```

Out[70]:
```
'X variable value is: 55 test for f-strings Y variable value is : 66'
```

In [71]:
```python
list(range(1,10))
```

Out[71]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [73]:
```python
name='Govardhan'
len(name)
# items - no of items (n-1) = 100(99)
# if u are using range (min) and max - items
```

Out[73]:
```
9
```

Python indexes

- left to right 0 to n
- right to left -1 to -n
- indexes max value (n-1)

In [0]:
```python
100 (99),1000(999)
```

format() function

- `str.format()` is one of the string formatting methods in Python3, which allows multiple substitutions and value formatting. This method lets us concatenate elements within a string through positional formatting.
- **Syntax** : `{ } .format(value)`

  Parameters : `(value)` : Can be an integer, floating point numeric constant, string, characters or even variables.

  Returntype : Returns a formatted string with the value passed as parameter in the placeholder position.

- The placeholders can be identified using named indexes {price}, numbered indexes {0}, or even empty placeholders {}.

In [75]:
```python
name="Govardhan"
age=23
print('my name is {b} and my age is : {a} '.format(a=age,b=name))
```

```
my name is Govardhan and my age is : 23
```

In [77]:
```python
name='Krishna'
age=25
sal=2000
commission=200
total_salary= sal+commission
print('My Name is : {2} And My Age is : {0} , my total salary is : {1}'.format(age,total_salary,name))
```

```
My Name is : Krishna And My Age is : 25 , my total salary is : 2200
```

In [80]:
```python
# empty place holders and it will occupy based on position based left to right
print('My Name is {} and I am living in {} '.format('Govardhan','Hyderabad'))
```

```
My Name is Govardhan and I am living in Hyderabad
```

In [81]:
```python
#index value based placeholders
print("this is a {2} sample {1} format {0} function usage example ".format(55,66,77))
```

```
this is a 77 sample 66 format 55 function usage example
```

In [82]:
```python
print("I want to work in {a} and i like programming {b} ".format(a='IT',b='Python '))
```

```
I want to work in IT and i like programming Python
```

In [83]:
```python
# the str.format() method
# using format option we can pass values using curly brackets {}
print ("{}    function example .".format("FORMAT-STRIN G"))
# using format option for a  value stored in a variable
str = "Sample text printing using format function in {}"
print (str.format("Python"))
```

```
FORMAT-STRIN G    function example .
Sample text printing using format function in Python
```

In [84]:
```python
age=23
name='Govardhan'
# formatting a string using a numeric constant
print("My Name is {} & I'm {} years old".format(name,age))
```

```
My Name is Govardhan & I'm 23 years old
```

In [85]:
```python
print("My Name is {a} & I'm {b} years old".format(a=name,b=age))
```

```
My Name is Govardhan & I'm 23 years old
```

In [88]:
```python
print('i like {0} programming and {1}  '.format('Python','Data Engineer'))
```

```
i like Python programming and Data Engineer
```