**Variables in Python**

**Variable:**

Named memory location.

or

Setting identities to memory location (reserved memory) to store and process information.

Syntax:

Identity = value

Ex:

A = 10

B = 34.56

C = “Python”

* Generally Memory allocation of a variable depends of type of variable. For example in C, int occupies 2 bytes memory.
* Python doesn’t support data types, hence it allocates memory depends on the value of variable.
* Therefore, by assigning different data types to variables, you can store integers, decimals, characters or strings in these variables.
* Python is completely object oriented, and not "statically typed".
* You do not need to declare variables before using them, or declare their type. Every variable in Python is an object.

**Example code:**

>>> i=10

>>> f=34.56

>>> s='python'

>>> print(i)

10

>>> print(i,f)

10 34.56

>>> print(i,f,s)

10 34.56 python

**Multiple Assignment :**

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

>>> a,b,c = 10,20,30

>>> print(a)

10

>>> a, b, c = 10 , 34.56, 'python'

>>> print(c)

python

>>> a=10 , b=20

SyntaxError: can't assign to literal

>>> a, b, c = 10

TypeError: 'int' object is not iterable

**Strings are defined either with a single quote or a double quotes :**

Strings can be represented either by using single quotes or double quotes.

In C, C++ or Java :

single quotes ---> character --> char ch = 'a' ;

double quotes --> String (set of characters) --> String s = "Java"

In Python :

ch = 'a' --> Single character string

s = 'Python' --> Multi character string

Note: Python doesn't support character type.

>>> s='python'

>>> print(s)

python

>>> s="python"

>>> print(s)

python

* Using two types of quotations is useful while writing following types of strings.
* Using double quotes makes it easy to include apostrophes (whereas these would terminate the string if using single quotes)

Output: This 'Python tutorial' makes you perfect

>>> 'This 'Python tutorial' makes you perfect'

SyntaxError: invalid syntax

>>> "This 'Python tutorial' makes you perfect"

"This 'Python tutorial' makes you perfect"

>>> msg = "This 'python tutorial' makes you perfect"

>>> print(msg)

This 'python tutorial' makes you perfect

>>> msg = 'This "python tutorial" makes you perfect'

>>> print(msg)

This "python tutorial" makes you perfect

>>> msg = "This "Python tutorial" makes you perfect"

SyntaxError: invalid syntax

>>> msg = "Python's strings tutorial"

>>> print(msg)

Python's strings tutorial

>>> msg = 'Python's string tutorial'

SyntaxError: invalid syntax

Escape character:

\n --> New line character

\t --> Tab space

\\ --> \

\' --> '

\" --> "

It is recommended to use escape characters(\) while working with strings:

>>> msg = "This \'python tutorial\' makes you perfect"

>>> print(msg)

This 'python tutorial' makes you perfect

>>> msg = "This \"python tutorial\" makes you perfect"

>>> print(msg)

This "python tutorial" makes you perfect

>>> msg = "Python\'s strings tutorial"

>>> print(msg)

Python's strings tutorial

>>> msg = 'Python\'s string tutorial'

>>> print(msg)

Python's string tutorial

**Try this msg:**

Output should print like 🡪 Annie said, “That’s the good one”

Code:

**Concatenate Variables:**

* We cannot concatenate different data types like string and number together.
* For example, we will concatenate "Java" with number "2".
* Unlike Java, which concatenates number with string without declaring number as string,
* Python requires declaring the number as string otherwise it will show a TypeError

>>> a = 10

>>> b = 20

>>> a+b

30

>>> a = 'Java'

>>> b = 'Python'

>>> a+b

'JavaPython'

>>> a = "Java"

>>> b = "2"

>>> c = "Python"

>>> print(a+b+c)

Java2Python

>>> a = "Java"

>>> b = 2

>>> c = "Python"

>>> print(a+b+c)

TypeError: Can't convert 'int' object to str implicitly

Type casting:

* Conversion of data from one type to another.
* Type casting will be done at runtime.
* Casting in python using pre-defined functions.

str() : Used to convert Integer into String

Used to convert Float into String

>>> a = 10

>>> b = 34.56

>>> a+b

44.56

>>> str(a)+str(b)

'1034.56'

int() :

It is used to convert String to Integer

>>> a = '10'

>>> a = '10' # string

>>> b = '20' # string

>>> a+b # concatenate

'1020'

>>> int(a) + int(b) # add

30

Note: In conversion process, String must be in integer form

>>> a = 'python'

>>> int(a)

ValueError: invalid literal for int() with base 10: 'python'

It is used to convert Float to Integer

>>> b = float(a)

>>> print(b)

10.0

>>> a = 34.56

>>> print(a)

34.56

>>> b = int(a)

>>> print(b)

34

>>> s = "100"

>>> print(s)

100

>>> i = int(s)

>>> print(i)

100

>>> s = "34.56"

>>> print(s)

34.56

>>> int(s)

ValueError: invalid literal for int() with base 10: '34.56'

>>> s = "python"

>>> int(s)

Traceback (most recent call last):

File "<pyshell#68>", line 1, in <module>

int(s)

ValueError: invalid literal for int() with base 10: 'python'

Convert int into char :

Chr() function converts integer into character.

Convert char into int :

Ord() function converts character into interger.

>>> b = True

>>> int(b)

1

>>> b = False

>>> int(b)

0

>>> i = 0

>>> bool(i)

False

>>> i = 1

>>> bool(i)

True

>>> f = 0.0

>>> bool(f)

False

>>> f = 12.34

>>> bool(f)

True

>>> s = "python"

>>> b = bool(s)

>>> print(b)

True

>>> s = 10

>>> b = bool(s)

>>> print(b)

True

>>> s = ''

>>> b = bool(s)

>>> print(b)

False

Basic programs using functions:

Code-1:

def fun():

print("fun....")

return

fun() # calling the function

Code-2:

def f1():

print("f1....")

return

def f2():

print("f2....")

return

f2() # calling the function

f1()

code:

def f1():

print("f1....")

return

def f2():

f1() # calling f1 function

print("f2....")

return

f2() # calling the function

>>> def f1():

print("f1.....")

return

>>> def f2():

print("f2.....")

return

>>> f1()

f1.....

>>> f2()

f2.....

>>> f1() , f2()

f1.....

f2.....

(None, None)

>>> def f2():

print("f2....")

return 10

>>> f1() , f2()

f1.....

f2....

(None, 10)

>>> f2()

f2....

10

>>> f1()

f1.....

Passing parameters:

def add(x, y):

z = x+y

return z

def mul(a, b):

c = a\*b

return c

res = add(10,20)

print("Sum : ", res)

res = mul(10,20)

print("Product : ", res)

**Taking input from the end-user:**

Input():

* Python library providing pre-defined method to take input from the end user.
* Input() function takes the input in string form.
* Once we collected in String format, we can convert into integer or float using conversion methods.

def printName(name):

print("My name is : ",name)

return

n = input("Enter name : ")

printName(n)

**We cannot process integer data without conversion:**

def add(x, y):

z = x+y

return z

x = input("Enter x value : ") # returns input in String form

y = input("Enter y value : ")

res = add(x,y)

print("Sum value is : ", res)

**Reading integer values:**

def add(x, y):

z = x+y

return z

s1 = input("Enter x value : ") # returns input in String form

s2 = input("Enter y value : ")

x = int(s1) #converts String into Integer

y = int(s2)

res = add(x,y)

print("Sum value is : ", res)

**Local & Global Variables:**

* Declaration of variable inside a function or block is referred as local variable.
* In Python when you want to use variable for complete program or module you declare it global variable.
* If you want to use the variable in a specific function or method you use local variable.

**Example Code:**

# declare variable and initialize

f = 101 ;

print f

def someFunction():

f = "I am in Java2Python"

print f

someFunction()

print f

* Variable "f" is global in scope and is assigned value 101 which is printed in output
* Variable f is again declared in function and assumes local scope.
* It is assigned value "I am learning Python." which is printed out as an output.
* This variable is different from the global variable "f" define earlier
* Once the function call is over, the local variable f is destroyed. When we again, print the value of "f" outside to function, it displays the value of global variable f=101

**output:**

101

I am in Java2Python

101

* Using the keyword global, you can reference the a global variable inside a function

**Code:**

# Access local variable from function

f = 101 ;

print f

def doAccess() :

global f

print f

f = "changing global variable value"

doAccess()

print f

* Variable "f" is global in scope and is assigned value 101 which is printed in output
* Variable f is declared using the keyword global. This is NOT a local variable but the same global variable declared earlier. Hence when we print its value the output is 101
* We changed the value of "f" inside the function. Once the function call is over, the changed value of the variable "f" persists. At line 12, when we again, print the value of "f" is it displays the value "changing global variable"

**Output:**

101

101

changing global variable value

Code-1 :

a = 30

def first():

print(a)

return

def second():

a = 20

print(a)

return

first()

second()

Code-2 :

a = 30

def first():

global a

a = 40

print(a)

return

def second():

# a = 20

print(a)

return

first()

second()

Code – 3 :

a = 30

def first():

a = 20

print(a)

global a

a = 40

print(a)

return

def second():

# a = 20

print(a)

return

first()

second()

**Delete a variable :**

* You can also delete variable using the command **del** "variable name"
* In the example below, we deleted variable f and when we proceed to print it, we get error "**variable name is not defined**" which means you have deleted the variable.

f = 101;

print f

del f

print f

**Rules to create python variables:**

* Variables names must start with a letter or an underscore, such as:

**\_underscore**

**underscore\_**

* The remainder of your variable name may consist of letters, numbers and underscores.

password1

n00b

un\_der\_scores

* Names are case sensitive.

max, MAX, mAx and MaX are each a different variable.

* Can be any (reasonable) length
* There are some reserved words which you cannot use as a variable name because Python uses them for other things.

**Convention Rules:**

* Readability is very important. Which of the following is easiest to read? I’m hoping you’ll say the first example.

python\_puppet

pythonpuppet

pythonPuppet

* Descriptive names are very useful. If you are writing a program that adds up all of the bad puns made in this book, which do you think is the better variable name?

total\_bad\_puns

super\_bad

* Avoid using the lowercase letter ‘l’, uppercase ‘O’, and uppercase ‘I’. Why? Because the l and the I look a lot like each other and the number 1. And O looks a lot like 0.