**Variables**

* Variables are nothing but reserved memory locations to store values
* Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory
* We don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.
* Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.
* It is recommended to use lowercase letters for the variable name. Rahul and rahul both are two different variables.
* also known as an identifier

# Variable Naming

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program. The rules to name an identifier are given below.

* The first character of the variable must be an alphabet or underscore ( \_ ).
* All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore, or digit (0-9).
* Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, \*).
* Identifier name must not be similar to any keyword defined in the language.
* Identifier names are case sensitive; for example, my name, and MyName is not the same.
* Identifier can be of any length.

Examples of valid identifiers: a123, \_n, n\_9, etc.

Examples of invalid identifiers: 1a, n%4, n 9, etc.

# Declaring Variable and Assigning Values

We don't need to declare explicitly variable in Python. When we assign any value to the variable, that variable is declared automatically.

The equal (=) operator is used to assign value to a variable.

# Variables Definition and Initialization

* Python variables do not need explicit declaration to reserve memory space
* The declaration happens automatically when you assign a value to a variable
* You can change the value of a variable in your program at any time, and Python will always keep track of its current value.

#!/usr/bin/python

Counter = 100 # An integer assignment

miles = 1000.0 # A floating point

name = "John" # A string

print counter

print miles

print name

Output:

100

1000.0

John

# Multiple Assignment

Python allows multiple assignments

Single value to several variables simultaneously

a = b = c = 1

1. Multiple value to multiple variables simultaneously

a, b, c = 3, 2.56, "john"

#!/usr/bin/python

a = b = c = 1

print a

print b

print c

print "\n"

a, b, c = 3, 2.56, "john"

print a

print b

print c

Output:

1

1

1

3

2.56

john

# Global and Local Variables

**Local variables** are the variables that declared inside the function and have scope within the function.

**Global variables** can be used throughout the program, and its scope is in the entire program. We can use global variables inside or outside the function.

#!/usr/bin/python

# This function uses global variable s

def f():

print s

# Global scope

s = "I love Geeksforgeeks"

f()

Output:

I love Geeksforgeeks

#!/usr/bin/python

# This function has a variable with name same as s.

def f():

s = "Me too."

print s

# Global scope

s = "I love Geeksforgeeks"

f()

print s

Output:

Me too.

I love Geeksforgeeks

**what will happen, if we change the value of s inside of the function f()?**

#!/usr/bin/python

def f():

# This program will NOT show error if we comment below line.

print s

s = "Me too."

print s

# Global scope

s = "I love Geeksforgeeks"

f()

print s

Runtime Errors:

Traceback (most recent call last):

File "/home/580423c254cc0aef0e80d3b1ddec63fa.py", line 12, in <module>

f()

File "/home/580423c254cc0aef0e80d3b1ddec63fa.py", line 3, in f

print s

UnboundLocalError: local variable 's' referenced before assignment

## “global”

Any variable which is changed or created inside of a function is local, if it hasn’t been declared as a global variable. To tell Python, that we want to use the global variable, we have to use the keyword “global”,

We only need to use global keyword in a function if we want to do assignments / change them.

**global is not needed for printing and accessing. Why?**

Python “assumes” that we want a local variable due to the assignment to s inside of f(), so the first print statement throws this error message. Any variable which is changed or created inside of a function is local, if it hasn’t been declared as a global variable.

#!/usr/bin/python

def f():

global s

print s

s = "Me too."

print s

# Global scope

s = "I love Geeksforgeeks"

f()

print s

Output:

I love Geeksforgeeks

Me too.

Me too.

# Print Single and Multiple variable

**For multiple variable**:

* “print variable” prints the variables without any brackets ‘()’ and splitted by a space
* “print(variable)” prints the variables with brackets ‘()’ and splitted by a coma ‘,’ so it’s treated as a tuple

#!/usr/bin/python

print 1

print (1)

print ((1))

print "\n"

print 1, 2

print (1, 2)

print ((1, 2))

Output:

1

1

1

1 2

(1, 2)

(1, 2)

**Note:** In Python 3.0, the print statement is changed to print() function. Below are equivalent codes in Python 3.0.

#!/usr/bin/python

#print 1 # SyntaxError: Missing parentheses in call to 'print'

print (1)

print ((1))

print("\n")

#print 1, 2 # SyntaxError: Missing parentheses in call to 'print'

print (1, 2)

print ((1, 2))

Output:

1

1

1 2

(1, 2)

# Swap two variables in one line

In Python, there is a simple and syntactically neat construct to swap variables

“x, y = y, x”

#!/usr/bin/python

x = 3

y = 5

print x

print y

x, y = y, x

print x

print y

Output:

3

5

5

3

# Delete a variable

We can delete the variable using the del keyword. The syntax is given below.

del <variable\_name>

Example

# Assigning a value to x

x = 6

print(x)

# deleting a variable.

del x

print(x)

Output:

6

Traceback (most recent call last):

File "C:/Users/DEVANSH SHARMA/PycharmProjects/Hello/multiprocessing.py", line 389, in

print(x)

NameError: name 'x' is not defined

# Object References

Python is the highly object-oriented programming language; that's why every data item belongs to a specific type of class. Consider the following example.

print("John")

type("John")

Output:

John

<class 'str'>

Variables are a symbolic name that is a reference or pointer to an object. The variables are used to denote objects by that name.

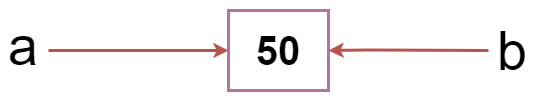
Example

a = 50



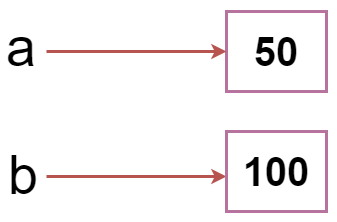
a = 50

b = a



a = 50

b =100



# Object Identity

In Python, every created object identifies uniquely in Python. Python provides the guaranteed that no two objects will have the same identifier. The built-in id() function, is used to identify the object identifier.

a = 50

b = a

print(id(a))

print(id(b))

# Reassigned variable a

a = 500

print(id(a))

print(id(b))

Output:

11095872

11095872

140010819366224

11095872

# Maximum Possible Value of an Integer in Python

Unlike the other programming languages, Python doesn't have long int or float data types. It treats all integer values as an int data type.

There is no limitation number by bits and we can expand to the limit of our memory.

Python doesn't have any special data type to store larger numbers.

# A Python program to display that we can store

# large numbers in Python

a = 10000000000000000000000000000000000000000000

a = a + 1

print(type(a))

print (a)

Output:

<class 'int'>

10000000000000000000000000000000000000000001

# Basic Fundamentals

This section contains the fundamentals of Python, such as:

1. Tokens and their types
2. Comments

## Tokens:

The tokens can be defined as a punctuator mark, reserved words, and each word in a statement.

The token is the smallest unit inside the given program.

There are following tokens in Python:

* Keywords
* Identifiers
* Literals
* Operators

# References

<https://www.tutorialspoint.com/python/python_variable_types.htm>

<https://www.javatpoint.com/python-variables>

# END OF FILE