VARIABLES, CONSTANTS, LITERALS AND DATA TYPES

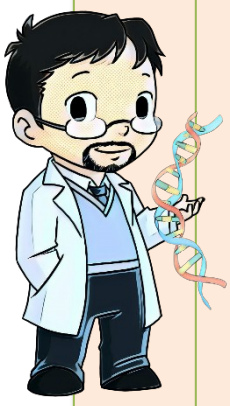
4

INTRODUCTION

The basis of a programming language contains data elements and the block in which they are being stored. Specific names are given for each of these, and special functionalities can be performed on them. In a programming language, they are called as Variables, Constants, and Literals. In this article, we will look at Python Constants, Variables, and Literals along with their types and examples.

Apart from those, you will also know the various data types that Python recognizes. Data types are important concepts in programming that cannot be missed out.

INTENDED LEARNING OUTCOMES



Upon the completion of this lesson, you should be able to:

1. define variables, constant, and literals in Python;
2. understand the various data types and use those accordingly; and
3. write simple Python program applying these concepts.

LESSON PROPER

PYTHON VARIABLES

A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data that can be changed later in the program. For example,

```
number = 10
```

Here, we have created a variable named number. We have assigned the value 10 to the variable.

You can think of variables as a bag to store books in it and that book can be replaced at any time.

```
number = 10
number = 1.1
```

Initially, the value of number was 10. Later, it was changed to 1.1.

Note: In Python, we don't actually assign values to the variables. Instead, Python gives the reference of the object(value) to the variable.

Assigning values to Variables in Python

As you can see from the above example, you can use the assignment operator = to assign a value to a variable.

Example 1: Declaring and assigning value to a variable

```
website = "apple.com"
print(website)
```

Output

```
apple.com
```

In the above program, we assigned a value apple.com to the variable website. Then, we printed out the value assigned to website i.e. apple.com

Note: Python is a type-inferred language, so you don't have to explicitly define the variable type. It automatically knows that apple.com is a string and declares the website variable as a string.

Example 2: Changing the value of a variable

```
website = "apple.com"
print(website)

# assigning a new value to website
website = "nipsc.edu.ph"

print(website)
```

Output

```
apple.com
nipsc.edu.ph
```

In the above program, we have assigned apple.com to the website variable initially. Then, the value is changed to nipsc.edu.ph.

Example 3: Assigning multiple values to multiple variables

```
a, b, c = 5, 3.2, "Hello"

print (a)
print (b)
print (c)
```

If we want to assign the same value to multiple variables at once, we can do this as:

```
x = y = z = "same"

print (x)
print (y)
print (z)
```

The second program assigns the same string to all the three variables x, y and z.

CONSTANTS

A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

You can think of constants as a bag to store some books which cannot be replaced once placed inside the bag.

Assigning value to constant in Python

In Python, constants are usually declared and assigned in a module. Here, the module is a new file containing variables, functions, etc which is imported to the main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example 3: Declaring and assigning value to a constant

Create a constant.py:

```
PI = 3.14
GRAVITY = 9.8
Create a main.py:
import constant

print(constant.PI)
print(constant.GRAVITY)
```

Output

```
3.14
9.8
```

In the above program, we create a constant.py module file. Then, we assign the constant value to PI and GRAVITY. After that, we create a main.py file and import the constant module. Finally, we print the constant value.

Note: In reality, we don't use constants in Python. Naming them in all capital letters is a convention to separate them from variables, however, it does not actually prevent reassignment.

Rules and Naming Convention for Variables and constants

1. Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_). For example:

```
snake_case
MACRO_CASE
camelCase
CapWords
```

2. Create a name that makes sense. For example, vowel makes more sense than v.

3. If you want to create a variable name having two words, use underscore to separate them. For example:

```
my_name
current_salary
```

4. Use capital letters possible to declare a constant. For example:

```
PI
G
MASS
SPEED_OF_LIGHT
TEMP
```

5. Never use special symbols like !, @, #, $, %, etc.

6. Don't start a variable name with a digit.

LITERALS

Literal is a raw data given in a variable or constant. In Python, there are various types of literals they are as follows:

Numeric Literals

Numeric Literals are immutable (unchangeable). Numeric literals can belong to 3 different numerical types: Integer, Float, and Complex.

Example 4: How to use Numeric literals in Python?

```
a = 0b1010     #Binary Literals
```

```
b = 100         #Decimal Literal
c = 0o310       #Octal Literal
d = 0x12c       #Hexadecimal Literal

#Float Literal
float_1 = 10.5
float_2 = 1.5e2

#Complex Literal
x = 3.14j

print(a, b, c, d)
print(float_1, float_2)
print(x, x.imag, x.real)
```

Output

```
10 100 200 300
10.5 150.0
3.14j 3.14 0.0
```

In the above program,

- We assigned integer literals into different variables. Here, a is binary literal, b is a decimal literal, c is an octal literal and d is a hexadecimal literal.
- When we print the variables, all the literals are converted into decimal values.
- 10.5 and 1.5e2 are floating-point literals. 1.5e2 is expressed with exponential and is equivalent to $1.5 * 10^2$.
- We assigned a complex literal i.e 3.14j in variable x. Then we use imaginary literal (x.imag) and real literal (x.real) to create imaginary and real parts of complex numbers.

String literals

A string literal is a sequence of characters surrounded by quotes. We can use both single, double, or triple quotes for a string. And, a character literal is a single character surrounded by single or double quotes.

Example 7: How to use string literals in Python?

```
strings = "This is Python"
char = "C"
multiline_str = """This is a multiline string with more than one line code."""
unicode = u"\u00dcnic\u00f6de"
raw_str = r"raw \n string"
```

```
print(strings)
print(char)
print(multiline_str)
print(unicode)
print(raw_str)
```

Output

```
This is Python
C
This is a multiline string with more than one line code.
Ünicöde
raw \n string
```

In the above program, This is Python is a string literal and C is a character literal.

The value in triple-quotes """ assigned to the multiline_str is a multi-line string literal.

The string u"\u00dcnic\u00f6de" is a Unicode literal which supports characters other than English. In this case, \u00dc represents Ü and \u00f6 represents ö.

r"raw \n string" is a raw string literal.

Boolean literals

A Boolean literal can have any of the two values: True or False.

Example 8: How to use boolean literals in Python?

```
x = (1 == True)
y = (1 == False)
a = True + 4
b = False + 10

print("x is", x)
print("y is", y)
print("a:", a)
print("b:", b)
```

Output

```
x is True
```

```
y is False
a: 5
b: 10
```

In the above program, we use boolean literal True and False. In Python, True represents the value as 1 and False as 0. The value of x is True because 1 is equal to True. And, the value of y is False because 1 is not equal to False.

Similarly, we can use the True and False in numeric expressions as the value. The value of a is 5 because we add True which has a value of 1 with 4. Similarly, b is 10 because we add the False having value of 0 with 10.

Special literals

Python contains one special literal i.e. None. We use it to specify that the field has not been created.

Example 9: How to use special literals in Python?

```
drink = "Available"
food = None

def menu(x):
    if x == drink:
        print(drink)
    else:
        print(food)

menu(drink)
menu(food)
```

Output

```
Available
None
```

In the above program, we define a menu function. Inside menu, when we set the argument as drink then, it displays Available. And, when the argument is food, it displays None.

Literal Collections

There are four different literal collections: List literals, Tuple literals, Dict literals, and Set literals.

Example 10: How to use literals collections in Python?

```
fruits = ["apple", "mango", "orange"] #list
numbers = (1, 2, 3) #tuple
alphabets = {'a':'apple', 'b':'ball', 'c':'cat'} #dictionary
vowels = {'a', 'e', 'i' , 'o', 'u'} #set

print(fruits)
print(numbers)
print(alphabets)
print(vowels)
```

Output

```
['apple', 'mango', 'orange']
(1, 2, 3)
{'a': 'apple', 'b': 'ball', 'c': 'cat'}
{'e', 'a', 'o', 'i', 'u'}
```

In the above program, we created a list of fruits, a tuple of numbers, a dictionary dict having values with keys designated to each value and a set of vowels.

DATA TYPES

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. There are various data types in Python. Some of the important types are listed below.

Python Numbers

Integers, floating point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python.

We can use the type() function to know which class a variable or a value belongs to. Similarly, the isinstance() function is used to check if an object belongs to a particular class.

```
a = 5
print(a, "is of type", type(a))
```

```
a = 2.0
```

```
print(a, "is of type", type(a))
a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

Output

```
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is complex number? True
```

Integers can be of any length, it is only limited by the memory available.

A floating-point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is an integer, 1.0 is a floating-point number.

Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are some examples.

```
>>> a = 1234567890123456789
>>> a
1234567890123456789
>>> b = 0.1234567890123456789
>>> b
0.12345678901234568
>>> c = 1+2j
>>> c
(1+2j)
```

Notice that the float variable b got truncated.

Python List

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ].

```
a = [1, 2.2, 'python']
```

We can use the slicing operator [ ] to extract an item or a range of items from a list. The index starts from 0 in Python.

```
a = [5,10,15,20,25,30,35,40]

# a[2] = 15
print("a[2] = ", a[2])

# a[0:3] = [5, 10, 15]
print("a[0:3] = ", a[0:3])

# a[5:] = [30, 35, 40]
print("a[5:] = ", a[5:])
```

Output

```
a[2] =  15
a[0:3] =  [5, 10, 15]
a[5:] =  [30, 35, 40]
```

Lists are mutable, meaning, the value of elements of a list can be altered.

```
a = [1, 2, 3]
a[2] = 4
print(a)
```

Output

```
[1, 2, 4]
```

Python Tuple

Tuple is an ordered sequence of items same as a list. The only difference is that tuples are immutable. Tuples once created cannot be modified.

Tuples are used to write-protect data and are usually faster than lists as they cannot change dynamically.

It is defined within parentheses () where items are separated by commas.

```
t = (5,'program', 1+3j)
```

We can use the slicing operator [] to extract items but we cannot change its value.

```
t = (5,'program', 1+3j)
```

```
# t[1] = 'program'
print("t[1] = ", t[1])

# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3])

# Generates error
# Tuples are immutable
t[0] = 10
```

Output

```
t[1] =  program
t[0:3] =  (5, 'program', (1+3j))
Traceback (most recent call last):
  File "test.py", line 11, in <module>
    t[0] = 10
TypeError: 'tuple' object does not support item assignment
```

Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

```
s = "This is a string"
print(s)
s = '''A multiline
string'''
print(s)
```

Output

```
This is a string
A multiline
string
```

Just like a list and tuple, the slicing operator [ ] can be used with strings. Strings, however, are immutable.

```
s = 'Hello world!'

# s[4] = 'o'
print("s[4] = ", s[4])
```

```
# s[6:11] = 'world'
print("s[6:11] = ", s[6:11])

# Generates error
# Strings are immutable in Python
s[5] ='d'
```

Output

```
s[4] =  o
s[6:11] =  world
Traceback (most recent call last):
  File "<string>", line 11, in <module>
TypeError: 'str' object does not support item assignment
```

Python Set

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

```
a = {5,2,3,1,4}

# printing set variable
print("a = ", a)

# data type of variable a
print(type(a))
```

Output

```
a =  {1, 2, 3, 4, 5}
<class 'set'>
```

We can perform set operations like union, intersection on two sets. Sets have unique values. They eliminate duplicates.

```
a = {1,2,2,3,3,3}
print(a)
```

Output

```
{1, 2, 3}
```

Since, set are unordered collection, indexing has no meaning. Hence, the slicing operator [ ] does not work.

```
>>> a = {1,2,3}
>>> a[1]
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
TypeError: 'set' object does not support indexing
```

Python Dictionary

Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type.

```
>>> d = {1:'value','key':2}
>>> type(d)
<class 'dict'>
```

We use key to retrieve the respective value. But not the other way around.

```
d = {1:'value','key':2}
print(type(d))

print("d[1] = ", d[1])

print("d['key'] = ", d['key'])

# Generates error
print("d[2] = ", d[2])
```

Output

```
<class 'dict'>
d[1] =  value
d['key'] =  2
Traceback (most recent call last):
  File "<string>", line 9, in <module>
KeyError: 2
```

Conversion Between Data Types

We can convert between different data types by using different type conversion functions like int(), float(), str(), etc.

```
>>> float(5)
5.0
```

Conversion from float to int will truncate the value (make it closer to zero).

```
>>> int(10.6)
10
>>> int(-10.6)
-10
```

Conversion to and from string must contain compatible values.

```
>>> float('2.5')
2.5
>>> str(25)
'25'
>>> int('1p')
Traceback (most recent call last):
  File "<string>", line 301, in runcode
  File "<interactive input>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1p'
```

We can even convert one sequence to another.

```
>>> set([1,2,3])
{1, 2, 3}
>>> tuple({5,6,7})
(5, 6, 7)
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

To convert to dictionary, each element must be a pair:

```
>>> dict([[1,2],[3,4]])
{1: 2, 3: 4}
>>> dict([(3,26),(4,44)])
{3: 26, 4: 44}
```

SUMMARY

- A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data that can be changed later in the program.
- You can use the assignment operator = to assign a value to a variable.
- A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.
- Rules and Naming Convention for Variables and constants
    - Constant and variable names should have a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
    - Create a name that makes sense.
    - If you want to create a variable name having two words, use underscore to separate them.
    - Use capital letters possible to declare a constant.
    - Never use special symbols like !, @, #, $, %, etc.
    - Don't start a variable name with a digit.
- Literal is a raw data given in a variable or constant.
- The following are Python data types:
    - Numbers
    - Lists
    - Tuples
    - String
    - Set
    - Dictionary

SELF-LEARNING ASSESSMENT

Let us see how much you have learned from this lesson. Answer the following questions.

ESSAY: Based on your understanding, discuss the following items using your own words. Each item is 10 points.

1. What are the differences between variables, constants and literals? Their similarities?

2. What is the difference between an int and a float?

ENRICHMENT ACTIVITY

In the lab, do the following:

You work in baseball operations at the NY Mets and are purchasing baseball caps from NewEra for next season. NewEra sells two types of caps, a snapback and a flex cap. The price points for each hat are listed below. Based on forecasted demand for next season you plan to buy 40,000 snapbacks and 70,000 flex hats.

- Flex=$12.45
- Snapback=$10.60

Create variables for: snap_num (the number of snap caps), flex_num (the number of flex caps), along with the corresponding prices: snap_px, flex_px.