TOPIC 4: Python Variables

Introduction.

Variables help programs become much more dynamic, and allow a program to always reference a value in one spot, rather than the programmer needing to repeatedly type it out, and, worse, change it if they decide to use a different definition for it.

Variables can be called just about whatever you want. You wouldn't want them to `conflict` with function names, and they also cannot start with a number.

Variables are containers for storing data values. Unlike other programming languages, Python has no command for declaring a variable. A variable is created the moment you first assign a value to it

Objectives

Objectives by the end of this topic you should be able to:

- Create variables
- Assign value to multiple variables
- Differentiate between global and local variables

Learning activities

Learning Activity 4.1: Reading

Read further on declaring local and global variables.

Learning Activity 4.2: Journal

Write a python program to demonstrate usage of different variables.

Learning Activity 4.3: Discussion

Write a Python program to check the sum of three elements (each from an array) from three arrays is equal to a target value. Print all those three-element combinations.

Assessment

Topic resources

1. The Python Tutorial¶. (n.d.). Retrieved from https://docs.python.org/3/tutorial/index.html
2. Mueller, J. P. (n.d.). *Beginning Programming with Python For Dummies*. S.l.: For Dummies.
3. (n.d.). Python 3.7.4 documentation. Retrieved from https://docs.python.org/3
4. (n.d.). Git Handbook. Retrieved from https://guides.github.com/introduction/git-handbook/

5. Shaw, Z. (2017). *Learn Python 3 the hard way: a very simple introduction to the terrifyingly beautiful world of computers and code*. Boston: Addison-Wesley.
6. Bader, D. (2018). *Python tricks: the book*. Vancouver, BC: Dan Bader.
7. Downey, A. B. (2015). *Think Python*. Sebastopol: OReilly.
8. Ramalho, L. (2016). *Fluent Python:*Beijing: OReilly.

URL Links

https://www.tutorialspoint.com/python3/python_variable_types.htm

https://www.geeksforgeeks.org/global-local-variables-python/

https://www.geeksforgeeks.org/python-scope-of-variables/?ref=rp

https://www.geeksforgeeks.org/private-variables-python/?ref=rp

https://www.geeksforgeeks.org/python-program-to-swap-two-variables/?ref=rp

TOPIC 4 NOTES

Python is not "statically typed". We do not need to declare variables before using them, or declare their type. A variable is created the moment we first assign a value to it.

```
#!/usr / bin / python

# An integer assignment
age = 45

# A floating point
salary = 1456.8

# A string
name = "John"

print(age)
print(salary)
print(name)
```

Output:

```
45

1456.8

John
```

Rules for creating variables in Python are same as they are in other high-level languages. They are:

a) A variable name must start with a letter or the underscore character.
b) A variable name cannot start with a number.
c) A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _).
d) Variable names are case-sensitive (name, Name and NAME are three different variables).
e) The reserved words(keywords) cannot be used naming the variable.

Assigning a single value to multiple variables:
Also Python allows to assign a single value to several variables simultaneously.
For example:

```
#!/usr / bin / python

a = b = c = 10

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

Output:

```
10

10

10
```

Assigning a different values to multiple variables:

```
#!/usr / bin / python

a, b, c = 1, 20.2, "Python Programming"

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

Output:

```
1

20.2

GeeksforGeeks
```

Can we use same name for different types?
If we use same name, the variable starts referring to new value and type.

```
#!/usr / bin / python

a = 10
a = "Python Programming"

print(a)
```

Output:

```
Python Programming
```

How does + operator work with variables?

```
#!/usr / bin / python

a = 10
b = 20
print(a+b)

a = "Programming in"
b = "Python"
print(a+b)
```

Output:

```
30

Programming in Python
```

Can we use + for different types also?

No using for different types would produce error.

```
#!/usr / bin / python

a = 10
b = "Python"
print(a+b)
```

Output :

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Creating objects (or variables of a class type):

Please refer Class, Object and Members for more details.

```
# Python program to show that the variables with a value
# assigned in class declaration, are class variables and
# variables inside methods and constructors are instance
# variables.

# Class for Computer Science Student
class CSStudent:

    # Class Variable
    stream = 'cse'

    # The init method or constructor
    def __init__(self, roll):

        # Instance Variable
        self.roll = roll

# Objects of CSStudent class
a = CSStudent(101)
b = CSStudent(102)

print(a.stream)  # prints "cse"
print(b.stream)  # prints "cse"
print(a.roll)    # prints 101

# Class variables can be accessed using class
# name also
print(CSStudent.stream) # prints "cse"
```

Output:

```
cse

cse

101

cse
```

Global and Local Variables in Python

Global variables are the one that are defined and declared outside a function and we need to use them inside a function

```
# This function uses global variable s
def f():
    print(s)

# Global scope
s = "I love programming in python"
f()
```

Output:

```
I love programming in Python
```

If a variable with same name is defined inside the scope of function as well then it will print the value given inside the function only and not the global value.

```
# This function has a variable with
# name same as s.
def f():
    s = "Me too."
    print(s)

# Global scope
s = "I love programming in python"
f()
print(s)
```

Output:

```
Me too.
I love programming in Python.
```

The variable s is defined as the string “I love programming in Python”, before we call the function f(). The only statement in f() is the “print s” statement. As there is no local s, the value from the global s will be used.

The question is, what will happen, if we change the value of s inside of the function f()? Will it affect the global s as well? We test it in the following piece of code:

```
def f():
    print(s)

    # This program will NOT show error
    # if we comment below line.
    s = "Me too."

    print(s)

# Global scope
s = "I love programming in Python"
f()
print(s)
```

Output:

```
Line 2: undefined: Error: local variable 's' referenced before assignment
```

To make the above program work, we need to use “global” keyword. 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. To tell Python, that we want to use the global variable, we have to use the keyword **“global”**, as can be seen in the following example:

```
# This function modifies global variable 's'
def f():
    global s
    print(s)
    s = "Look for Geeksforgeeks Python Section"
    print(s)

# Global Scope
s = "Python is great!"
f()
print(s)
```

Now there is no ambiguity.
Output:

```
Python is great!
Look for programming in Python Section.
Look for programming in Python Section.
```

A good Example

```
a = 1

# Uses global because there is no local 'a'
def f():
    print('Inside f() : ', a)

# Variable 'a' is redefined as a local
def g():
    a = 2
    print('Inside g() : ',a)

# Uses global keyword to modify global 'a'
def h():
    global a
    a = 3
    print('Inside h() : ',a)

# Global scope
print('global : ',a)
f()
print('global : ',a)
g()
print('global : ',a)
h()
print('global : ',a)
```

Output:

```
global :  1
Inside f() :  1
global :  1
Inside g() :  2
global :  1
Inside h() :  3
global :  3
```

Revision questions

1. What is the output of the following code

```
x = 50
def fun1():
    x = 25
    print(x)
fun1()
print(x)
```

2. What is the output of the following code

```
def func1():
    x = 50
    return x
func1()
print(x)
```

3. Select all the valid String creation in Python
    - ☐ str1 = “str1”
    - ☐ str1 = ‘str1’
    - ☐ str1 = ”’str1”’
    - ☐ str1 = str(“str1”)
4. Write a Python program to iterate over an enum class and display individual member and their value. Go to the editor
   Expected Output:
   Afghanistan = 93
   Albania = 355
   Algeria = 213
   Andorra = 376
   Angola = 244
   Antarctica = 672
   Click me to see the sample solution