Variables and Simple Data Types

BY DR. SABEEHUDDIN HASAN

Running hello_world.py

- When you run a simple program hello_world.py
  - >>> print("Hello Python world!")
- You see the following output
  - Hello Python world!
- **Variables**
  - Add a new line at the beginning of the file, and modify the second line
    - message = "Hello Python world!"
    - print(message)
  - You see the same output
    - Hello Python world!

Variables

- Add two new lines of code
  - **message = "Hello Python world!"**
  - **print(message)**
  - **message = "Hello Python Crash Course world!"**
  - **print(message)**
- Now when you run *hello_world.py*, you should see two lines of output:
  - **Hello Python world!**
  - **Hello Python Crash Course world!**

Naming and Using Variables

- Variable names can contain only letters, numbers, and underscores
  - They can start with a letter or an underscore, but not with a number
  - Valid name: *message_1,* invalid: *1_message*
- Spaces are not allowed, underscores can be used
  - *New_message* is correct, *New message* is invalid
- Avoid using Python keywords and function names as variable names
  - E.g. *print* as a variable name is invalid
- Variables should be descriptive
  - Better to write new_message, rather than n_m
- Python is case sensitive

Avoiding Naming Errors When Using Variables

- Consider the following code

```
message = "Hello Python Crash Course reader!"
print(mesage)
```

- When an error occurs, Python helps you trace the issue by traceback
- A *traceback* is a record of where the interpreter ran into trouble when trying to execute your code

```
Traceback (most recent call last):
1 File "hello_world.py", line 2, in <module>
2 print(mesage)
^^^^^^
3 NameError: name 'mesage' is not defined. Did you mean: 'message'?
```

Avoiding Naming Errors When Using Variables

- The Python interpreter doesn't spellcheck your code
  - But ensures that variable names are spelled consistently

```
mesage = "Hello Python Crash Course reader!"
print(mesage)
```

  - In this case, the program runs successfully!

```
Hello Python Crash Course reader!
```

Variables Are Labels - Strings

- In Python think of variables as labels that you can assign to values
- A variable references a certain value
- **Strings**
  - A *string* is a series of characters.
  - Anything inside quotes is considered a string in Python
  - You can use single or double quotes around your strings like this

    "This is a string."

    'This is also a string.'
  - This flexibility allows you to use quotes and apostrophes within your strings

    'I told my friend, "Python is my favorite language!"'

    "The language 'Python' is named after Monty Python, not the snake."

Variables Are Labels - Strings

- ***Changing Case in a String with Methods***

  name = “muhammad ali"

  print(name.title())

  - Run this code and the following output is generated

    Muhammad Ali

  - Similarly one can change to upper or lower case

    name = “Muhammad Ali"

    print(name.upper())

    print(name.lower())

  - The output is as follows

    MUHAMMAD ALI

    muhammad ali

Variables Are Labels - Strings

- ***Using Variables in Strings***
  - In some situations, you'll want to use a variable's value inside a string
  - You might want to use two variables to represent a first name and last name and combine them to show full name

```
first_name = "Muhammad"
last_name = "Ali"
full_name = f"{first_name} {last_name}"
print(full_name)
```

  - These strings are called *f-strings, f* is for format
  - Output: **Muhammad Ali**

Variables Are Labels - Strings

- See the following code

```
first_name = "muhammad"
last_name = "ali"
full_name = f"{first_name} {last_name}"
print(f"Hello, {full_name.title()}!")
```

- Output: **Hello, Muhammad Ali!**
- In the above example try with
  - **message = f"Hello, {full_name.title()}!"**
  - **print(message)**

Variables Are Labels - Strings

- ***Adding Whitespace to Strings with Tabs or Newlines***
  - To add a tab to your text, use the character combination \t:
    - **>>> print("\tPython")**

      Python
  - To add a newline in a string, use the character combination \n:
    - **>>> print("Languages:\nPython\nC\nJavaScript")**

    Languages:

    Python

    C

    JavaScript

Variables Are Labels - Strings

- ***Stripping Whitespace***
  - *rstrip()* strips whitespace from the right of the string
  - >>> **favorite_language = 'python ‘**
  - >>> **favorite_language**

  ‘python ’
  - >>> **favorite_language.rstrip()**

  'python‘
- Try using functions lstrip() and strip()

Variables Are Labels - Strings

- ***Removing Prefixes***
  - Consider a URL with the common prefix *https://*.
  - We can remove this prefix by using the function *removeprefix*

    >>> **nostarch_url = 'https://nostarch.com'**

    >>> **nostarch_url.removeprefix('https://')**

    'nostarch.com‘
- ***Avoiding Syntax Errors with Strings***
  - A *syntax error* occurs when Python doesn’t recognize a section of your program as valid Python code
  - For example: Using an apostrophe within single quotes will produce an error

Variables Are Labels - Strings

- ***Avoiding Syntax Errors with Strings***
  - >>> **message = 'One of Python's strengths is its diverse community.'**
  - >>> **print(message)**

  - File "apostrophe.py", line 1
  - message = 'One of Python's strengths is its diverse community.'
  - ^
  - SyntaxError: unterminated string literal (detected at line 1)

Numbers

- Numbers are used quite often in programming to
  - keep score in games,
  - represent data in visualizations,
  - store information in web applications, and so on
- ***Integers***
  - You can add (+), subtract (-), multiply (*), and divide (/) integers in Python

    >>> **2 + 3** **Output:** 5

    >>> **3 – 2** **Output:** 1

    >>> **2 * 3** **Output:** 6

    >>> **3 / 2** **Output:** 1.5

Numbers

- Python uses two multiplication symbols to represent exponents
  - >>> **3 ** 2**
  - 9
  - >>> **3 ** 3**
  - 27
- You can also use parentheses to modify the order of operations
  - >>> **2 + 3*4**
  - 14
  - >>> **(2 + 3) * 4**
  - 20

Numbers

- **Floats**
  - Any number with a decimal point is a *float*
  - A decimal point can appear at any position in a number
  - >>> **0.1 + 0.1**
  - 0.2
  - >>> **2 * 0.1**
  - 0.2

Numbers

- ***Integers and Floats***
  - When you divide any two numbers, even if they are integers resulting in a whole number, you'll always get a float
  - >>> **4/2** **Output:** 2.0
  - Mixing an integer and a float results in a float
  - >>> **1 + 2.0** **Output:** 3.0
  - >>> **2 * 3.0** **Output:** 6.0
- ***Underscores in Numbers***
  - For long numbers, use underscores for readability
  - >>> **universe_age = 14_000_000_000**
  - >>> **print(universe_age)**
  - 14000000000

Numbers

- ***Multiple Assignment***
  - You can assign values to more than one variable using just a single line of code
  - >>> **x, y, z = 0, 0, 0**
- ***Constants***
  - A *constant* is a variable whose value stays the same throughout the life of a program
  - Python doesn't have built-in constant types
  - Python programmers use all capital letters to indicate a variable as a constant
  - MAX_CONNECTIONS = 5000

Boolean

- A *Boolean value* is either True or False
- It is like the value of a conditional expression after it has been evaluated
  - **>>> editable = True**
  - **>>> game_active = False**
  - **>>> print (editable)**
  - **>>> print (game_active)**
  - True
  - False

Complex

- Not only real numbers, Python can also handle complex numbers and its associated functions using the file “cmath”
- An complex number is represented by “**x + yi**”
- The real part can be accessed using **real** and imaginary part can be represented by **imag**

```
import cmath

z = 5 + 2j

# printing real and imaginary part of complex number

print("The real part of complex number is:", z.real)

print("The imaginary part of complex number is:", z.imag)
```

The real part of complex number is: 5.0

The imaginary part of complex number is: 2.0

Datatypes

- Python uses *data types* to categorize values in memory
- When an integer is stored in memory, it is classified as an int
- When a real number is stored in memory, it is classified as a float

```
intData = 4
floatData = 4.0
stringData = 'abc'
booleanData = True
print (type(intData), type(floatData), type (stringData), type(booleanData))
```

<class 'int'> <class 'float'> <class 'str'> <class 'bool'>

Comments

- Comments are indicated by a hash mark (#)
  - **#This is a comment**
  - **print ('Hello')**
  - Hello
- Multiline comment
  - **''' This is a multiline comment**
  - **It spans multiple lines**
  - **as shown'''**
  - **print ('Hello')**
  - Hello