Review Python Programming Features

CS313E - Elements of Software Design

Kia Teymourian

Table of contents

Python Programming Language

- Python (`https://www.python.org/`)is a simple but powerful scripting language.
- It was developed by **Guido van Rossum** in the Netherlands in the late 1980s.
- The language takes its name from the British comedy series Monty Python's Flying Circus.
- **Python is an interpreted language unlike C or Fortran that are compiled languages**.
- Python is concise and compact, and it is widely used.

Python Programming Language

- Python (`https://www.python.org/`)is a simple but powerful scripting language.
- It was developed by **Guido van Rossum** in the Netherlands in the late 1980s.
- The language takes its name from the British comedy series Monty Python's Flying Circus.
- **Python is an interpreted language unlike C or Fortran that are compiled languages**.
- Python is concise and compact, and it is widely used.

You can run python in different ways:

- Python has two basic modes: script and interactive.
- You can use both of these methods.
- Python Jupyter Notebook (`https://jupyter.org/`) is similar to command line interactive shell.

Python Programs

- We run the complex programs as python scripts.
- In Python, variable names are case sensitive.
- A single statement only goes as far as the end of the line. If you intend to continue on the next line you must use the continuation back slash symbol at the end of the line.
- **Indentation matters in Python.** You use indentation to delineate blocks of code that go together. As a rule of thumb use 2 spaces (instead of tabs) for indentation otherwise you will be scrolling horizontally to read your code.

main() function

- For complex problems, you will break up the task of solving the problem into separate subtasks.
- Each of the subtasks will be implemented as one or more functions. There will be a main function that will call these other functions to implement the solution. Even for the simplest problems, where you do not have any auxiliary functions, write the main() function and call it.

```
def main():
    ...
    ...

main()
```

Listing 1: Skeleton of your Python code with main function

Variables in Python

- Python uses the traditional ASCII character set.
- The latest version (3.8.3) also recognizes the Unicode character set.

Variable Identifiers:

Python is case sensitive.

These are rules for creating an identifiers:

- Must start with a letter or underscore (_)
- Can be followed by any number of letters, digits, or underscores.
- Cannot be a reserved word which are the following:

and,as,assert,break,class,continue,def,del,elif,
else,except,exec,finally,for,from,global,if,import,in,
is,lambda,not,or,pass,print,raise,return,try,while,with,yield

Variables

A variable in Python denotes a memory location.

- In that memory location is the address of where the actual value is stored in memory. Consider this assignment: `x = 1`
  A memory location is set aside for the variable x. The value 1 is stored in another place in memory. The address of the location where 1 is stored is placed in the memory location denoted by x.
- Later you can assign another value to x like so: `x = 2` In this case, the value 2 is stored in another memory location and its address is placed in the memory location denoted by x. The memory occupied by the value 1 is reclaimed by the ***garbage collector***.
- You may assign a **different type** to value to x like a floating point number.
  `x = 3.45`

Simple Input and Output

You can prompt the user to enter a value by using function input().

```
x = input ("Enter a number: ")
y = input ("Enter another number: ")
```

The print command allows you to print out the value of a variable.

```
print ('x = ', x)
print ("y = ", y)
```

Operators - Literal

A literal is a constant value for some of the built-in types. Here are some examples of literals.

```
>>> a = 1                    # 1 is an integer literal
>>> b = 2.3                  # 2.3 is a floating point literal
>>> c = False                # False is a boolean literal
>>> d = 5L                   # 5L is a long integer literal
>>> e = 8 + 6j               # 8 + 6j  is a complex literal
>>> f = "Hello"              # "Hello" is a string literal
```

Expression and Operators

Expression

An expression is composed of variables and operators. The simplest expression is just a variable. The value of an expression is evaluated before it is used.

Arithmetic Operators

These are the arithmetic operators that operate on numbers (integers or floats). The result of applying an arithmetic operator is a number.

- Addition: +
- Subtraction: -
- Multiplication: *
- Division: /
- Integer Division: //
- Remainder: %
- Exponentiation: **

Comparison Operators

There are 6 comparison operators. The result of applying the comparison operators is a Boolean - True or False.

- Equal to: ==
- Not equal to: !=
- Greater than: >
- Greater than or equal to: >=
- Less than: <
- Less than or equal to: <=

Boolean Operators

In Python, we have two Boolean literals - **True** and **False**.
But Python will also regard as False - the number zero (0), an empty string (""), or the reserved word None.
All other values are interpreted as True.
There are 3 Boolean operators (not, and, and or):

- **not**: unary operator that returns True if the operand is False and vice versa.
- **x and y**: if x is false, then that value is returned; otherwise y is evaluated and the resulting value is returned.
- **x or y**: if x is true, then that value is returned; otherwise y is evaluated and the resulting value is returned.

Bitwise Operators

The bitwise operators include the AND operator &, the OR operator |, the EXCLUSIVE OR operator $\wedge$ and the unary NOT operator ~.
The bitwise operators applies only to integer types.

Truth Table for EXCLUSIVE OR

a	**b**	$a \wedge b$
0	0	0
0	1	1
1	0	1
1	1	0

Binary Representation of an Integer

```
# bin() Return the binary representation of an integer.

>>> print(bin(84))
0b1010100

>>> print(0b1010100)
84

>>> print(bin(13))
0b1101

>>> print(0b1101)
13
```

Shift Operators

Shift operators are applied only to integer types.

- $x << k$: shift the bits in x, k places to the left.
  Multiplication by (2 ** k).
- $x >> k$: shift the bits in x, k places to the right filling in with the highest bit on the left hand side.
  Division by (2 ** k).

Example - Bitwise Operattions

```
a = 84              # 84 = 0101 0100
b = 13              # 13 = 0000 1101

print(a & b)        # 4 = 0000 0100
print(a | b)       # 93 = 0101 1101

print(a^b)         # 89 = 0101 1001

# Bitwise not operator: Returns one's complement of the number.
print(~a)             # -61 = 0011 1101

# Bitwise right shift
                    # 84 = 0101 0100
print(a >> 1)       # 42 = 0010 1010

# Bitwise left shift
                      # 84 = 0101 0100
print(a << 1)        # 168 = 1010 1000
```

Functions

- Python can be used as a functional programming language as well.
- You can think of a function as a small piece of code that has one specific functionality.
- Functions need to be called to be executed
- They can be called within main() or within other functions.
- Functions may or may not return a value or values.

The structure of a function definition is as follows:

```
def function_name ([formal_parameters]):
    ...
    body_of_the_function
    ...

    may_return_something
```

Pre-defined functions in Python.

```
x = -4.7
y = abs (x)      # y = 4.7
z = int (x)      # z = -4
```

You can find a complete list of built-in functions on python website
http://docs.python.org/library/functions.html
Other functions can be imported from modules before you can use them like string, math, and random.
You load these modules by using the import directive.

```
import string
import math, random
```

After you load the module you can call on the functions using the dot operator.

```
import math, random

x = 7
y = math.sqrt (x)
z = random.random()
```

User-defined Function

- One of the power of a programming language like python is that the user can define functions.
- To call a user defined function that is in the same program you simply call it by name without using the dot operator.

Try this code Example on the Google Colab:
`https://colab.research.google.com/drive/1hpxWygGHdvmoMcmGVhdxd3eMO_NIuq63?usp=sharing`