ECE326

PROGRAMMING LANGUAGES

Lecture 2 : Basics of the Python Programming Language

Kuei (Jack) Sun

ECE

University of Toronto

Fall 2020

Python

- High-level, general purpose programming language
- First released in 1991, by Guildo van Rossum
- Focuses on readability

Python

- Scripting Language
  - Convenience language for writing commands
  - Allows for rapid development of simple tools
  - E.g. Bash, Python, Perl
- Interpreted Language
  - Interpreter
    - Software which executes the program line-by-line at *runtime*
  - Most scripting languages comes with an interpreter
  - Generally much slower than compiled languages

Python

- Multi-paradigm
  - Procedural programming
    - Code is organized into modules, which contain functions
  - Object-oriented programming
    - Programmer defines classes
    - Classes are blueprints for creating objects
    - Objects have methods and attributes
    - Code is organized into classes
  - Functional programming
    - Code forms a tree of mathematical expressions

Python

- Two major versions
- Python2
  - First widely popular version of Python
  - Still used in many projects
  - Discontinued in 2020
- Python3
  - Released in 2008
  - *Not* backward-compatible
  - We will use Python 3.5 for this class

Interactive Mode

- Benefit of interpreted language
- Can execute code as you type

```
$ python3
Python 3.6.9 (default, Jul 21 2019, 14:33:59)
[GCC 7.4.0] on cygwin
Type "help", "copyright", "credits" or "license" for more
information.
>> x = 2
>> x + 3
5
>> exit()

$ …back in command prompt…
```

Source Code

- hello.py

```
#!/usr/bin/python3

# this is a comment (like // in C++)

# main is not required
print(“hello world!”)
```

Tells terminal which interpreter to run (we will be using Python3 for this course)

```
> python3 hello.py
hello world!
> chmod +x hello.py
> ls –l hello.py
-rwxr-xr-x 1 jack jack 41 Sep 10 12:10
hello.py
> ./hello.py
hello world!
```

Explicitly run with Python3

Make the script executable

Run script as executable

Value

- A unit of representation or data
  - Program can use or manipulate a value
  - Associated with a *type*
  - E.g. 5 is a value of type integer (integer value)

Type

- A set of values, and (implicitly) a set of behaviours on those values
  - A way to express *constraints*
  - E.g. you should only use integer instructions on integer types
- Conveys intended use
- Conveys *semantics* (meaning)
- Defines how values of that type is stored
  - E.g. Two’s complement for signed integers

Basic Types

- Usually correlates with machine data types
- Integers
  - E.g. `short, int, long` in C++
  - In Python

```
>> x = 5
>> type(x)
<class ‘int’>
```

- Floating-point numbers

```
>> type(3.14)
<class ‘float’>
```

- The term “primitive” types usually means basic types

Built-in Types

- Types natively supported by the programming language
- String

```
>> type(‘a’)
<class ‘str’>
```

There is no “char” type in Python, a character is just a string of length 1

- List

```
>> type([])
<class ‘list’>
```

You will learn about list, dictionary, and tuple in future lectures

- Dictionary

```
>> type({})
<class ‘dict’>
```

- Tuple

```
>> type(())
<class ‘tuple’>
```

Variable

- A name (i.e. *identifier*) associated with a *memory location* that contains a value
  - In Python, a variable is created through initialization

    ```
    >> foo = 6
    ```

    - In contrast, C++ requires variable declaration (e.g. `int a;`)
  - A variable has a type

    ```
    >> type(foo)
    <class ‘int’>
    ```

  - Name is used to refer to the stored value

    ```
    >> print(foo)
    6
    ```

Dynamic vs. Static Typing

- Statically Typed
  - Types of variables checked before runtime
  - Once declared, a variable may not change type
  - E.g. C++, Java
- Dynamically Typed
  - Types are checked at runtime, on the fly
  - A variable may change type
  - E.g. Python, Ruby

```
>> foo = 6
>> foo = “hello”   # this is allowed, foo now a string
```

Keyword

- A word reserved by the programming language
  - Has special meaning
  - Cannot be used as an identifier
- Common (between Python and C++)
  - *if, else, for, while, continue, break, return,* etc.
- Python only
  - *in, and, or, def, with, pass, yield,* etc.

Literal and Constant

- Literal: represents a fixed value in source code
  - E.g. 5, “`hello`”, `3.3`
- Constant is same as variable, except its value cannot be changed once initialized
  - Python does not enforce constants
    - Programmer enforced via naming convention
      - Use all uppercase letters, words separated by underscore
      - `MAX_LENGTH = 128`
  - C++ const keyword:

```
const int max_length = 128;
max_length = 64; // error: assignment of read-only variable
```

Expression

- A combination of one or more values, operators, and functions that produces another value
  - This process is known as *evaluation*

```
>> 3 + 5
8
>> -foo * math.sqrt(3)
-10.392304845413264
```

- An expression can have subexpressions
- Evaluation strategy
  - Discussed in later lectures

Operators

- Symbols or keywords that behave like functions
  - However, they differ syntactically and/or semantically
  - E.g. `add(1, 2)` vs. `1 + 2`
  - Follows operator precedence
  - Python uses same rules as mathematics (i.e. BEDMAS)
- Arithmetic
  - Same as C++, plus more…

```
>> 3**2       # 3 to the power of 2
9
>> 5/2        # true division
2.5
>> 5//2       # floor division
2
```

Operators

- Relational
  - Performs comparison, returns true or false
- Logical

C++	&&	||	!
Python	and	or	not

```
>> x = 5
>> y = 4
>> y + 1 == x and y < 5 or not print(“hello”)
True
```

  - Remember short-circuit evaluation?

Statement

- A syntactic unit of imperative programming
  - Performs “some” action(s)
  - May contain expressions
  - Does *not* evaluate to a value
    - If a statement is just an expression, the value is discarded/unused.

```
// In C++, this is valid, but
// doesn’t do anything (useful)
x + 5;
```

- Example
  - Loop control (i.e. continue and break)
  - Function return

Statement

- In C++
  - Terminates with a semicolon ;

```
int a = 1 + 2 + 3
        + 4 + 5 + 6 + 7 +
                  8 + 9 + 10;
```

- In Python
  - Terminates at end of line
  - Multi-line statement requires *line continuation*
    - Use backward slash \ to continue statement to next line

```
>> a = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + \
.. 9 + 10
55
>>
```

Compound Statement

- Contains one or more statements
- Control Flow
  - Execution results in choice made on which path to take
- Example
  - *If* statement
  - *switch* statement
  - *while* loop
  - *for* loop
  - Etc

Block

- Consists of one or more statements
- Usually part of control flow
  - E.g. Conditional in Python

```
if a == 3:
    pass
else:
    b = 5
    c = a + 6
d = a - 3
```

Same as an empty body in C++
Required for syntactic reason

Reminder:
Do *NOT* mix
tabs and
spaces

```
// in C++
if (a == 3) {}
else { … }
d = a – 3;
```

- Off-side rule
  - Blocks are expressed by their indentation

Conditional

- Python Syntax

```
if boolean-expression:
    block
elif boolean-expression:
    block
elif boolean-expression:
    block
…
else:
    block
```

Recall in C++:

```
if (bool-expr) {
      …
}
```

- No parentheses required for the conditions
- Python does not have a switch statement

Assignment

- Assignment is an *expression* in C++
  - Evaluates to the assigned value

```
int a, b;
a = b = 5;              // same as a = (b = 5);
if ((b = foo(a)))       // this is allowed
    assert(b != 0);     // always true
```

- Assignment is a *statement* in Python!

```
a = b = 5                       # syntax error, if statement
# same as below                 # expects an expression
temp = 5                        if a = 5:
a = temp                            b = 3
b = temp
```