Introduction to Python

Workshop Notes

Dr Amara Atif
amara.atif@uts.edu.au
FEIT

Topics to Cover

Keywords	Identifiers & Comments	Variables
Statement and Expression	Data types	Operators, Precedence, and Associativity

Keywords/Reserve words

- Python keyword is a unique programming term intended to perform some action
- They build the vocabulary of the Python language
- There are **36 keywords in Python in the 3.9.2 release**
  - Note that the list may change
  - The language could get away with some of the old keywords and bring in new ones in future releases

To get the up-to-date list, you can open Python shell and run the following commands as shown in the below snippet

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try',
'while', 'with', 'yield']
```

- All keywords in Python are **case sensitive**
- You CAN NOT use the keywords/reserve words for defining variables, functions or classes

Identifiers

Python Identifiers are **user-defined names used to represent a variable, function, class, or module**

Rules to form an identifier

- Use a **sequence of letters** either in **lowercase (a to z)** or **uppercase (A to Z)**
- You can also mix up **digits (0 to 9)** but **can not use digits to begin** an identifier name
- **No special characters from the keyboard** such as ['.', '!', '@', '#', '$', '%', '^', '&', '*' etc.] can be used an identifier name EXCEPT an **underscore (_)** is allowed (an underscore can be used in the start, middle, or end)
- The **keywords/reserve words cannot** be used as identifiers
- Identifiers names should be **meaningful** and **short**

Comments

- Comments in Python are the tagged lines of text to annotate a piece of code
- They are **non-executable part of the code**
- It is a good programming practice to always use comments as the code becomes self-evident and highly readable
- Types
  - Single-line comment
    - Begin with a hash (#) symbol and the white space character and continue to the end of the line
  - Multi-line comments
    - can be added by inserting a hash (#) symbol for each line or by using a multi-line string (triple quotes or docstring)

Single-line comment

```
# This is a sample Python script.
# print("Hello World!")
print("Hello World!") # First Python statement
Output:
Hello World!
```

Multi-line comments

```
# This is a sample Python script.
# The script prints a statement.
# print is a keyword in Python.
Output:
No output
```

```
"""
This is a sample Python script.
The script prints a statement.
print is a keyword in Python.
"""
print("Hello World!")
Output:
Hello World!
```

Variables

- A variable in Python represents an entity whose value can change as and when required
- Conceptually, it is a labelled memory location which holds the actual value and we can retrieve the value from our code by querying (using a reference) the entity
- Python has **two types** of variables
  - **Built-in variables** are the keywords of Python
  - **User-defined** variables are the variables created by the programmers as and when required

Rules for Using Variables

- User-defined variables naming convention follows all the rules for naming an identifier
- Variables do not require declaration (the data type)
  - ❑ You must initialise them before use
  - ❑ The declaration happens automatically when you assign a value to a variable
- The equal sign (=) is used to assign values to variables
- You can re-declare Python variables even after you have declared once
- In Python you cannot concatenate string with number directly, you need to declare them as a separate variable, and after that, you can concatenate number with string

```
# How to declare and use a variable "price".
price=10
print(price)
Output:
10
```

```
# Declare and use a variable "price".
price=10
print(price)
# Re-declare a variable even after it is declared once. It works fine.
price='dollars'
print(price)
Output:
10
dollars
```

```
# TypeError: different types cannot be combined
print("dollars"+10)
```

Output:

```
Traceback (most recent call last):
  File "C:\Users\Amara Atif\PycharmProjects\pythonProject-Learning\Test Code.py", line 2, in <module>
    print("dollars"+10)
TypeError: can only concatenate str (not "int") to str
```

```
# String concatenation and variable
a="Dollars"
b=10
print(a+str(b))
```

Output:

```
Dollars10
```

Scope of Variables

- **Global variables**
  - When you want to use the same variable for rest of your program, you declare it as a global variable
  - Global variables are usually declared at the start of the program
- **Local variables**
  - When you want to use the variable in a specific function or method, you use a local variable
  - Within a program, a local variable can have the same name as the global variable but the scope will be different

```
# Declare a global variable and use it
some_var= 10
print(some_var)
# Local variable in a function
def someFunction():
    some_var = 'I am learning Python'
    print(some_var)
# Once the function call is over, the local variable "some_var" is destroyed and now will use the value of the gloabl variable
someFunction()
print(some_var)
```

Output:

```
10
I am learning Python
10
```

Delete a Variable

The command **del** is used to delete a variable

```
# Declare a variable and use it
count=5;
print(count)
# Delete the variable
del count
print(count)
```

Output:

Test Code

```
"C:\Users\Amara Atif\PycharmProjects\pythonProject-Learning\venv\Scripts\python.exe" "C:/Users/Amara Atif/PycharmProjects/
5
Traceback (most recent call last):
  File "C:\Users\Amara Atif\PycharmProjects\pythonProject-Learning\Test Code.py", line 6, in <module>
    print(count)
NameError: name 'count' is not defined
```

Statement

- A statement in Python is a logical instruction which Python interpreter can read and execute
- It could be an:
  - Expression
  - Assignment statement
    - variable = expression (a=1)
    - if statement
    - for statement
    - while statement
    - import statements

Expression

- An expression is a combination of **operators** and **operands**
  - Need to be evaluated
  - If Python to print an expression, the interpreter **evaluates** the expression and displays the result
- Operators (+, -, *, /, +=, -=, *=, /= and so on)
  - Using expressions, we can perform operations like addition, subtraction, concatenation and so on
- Operands
  - Variables (x, y, z, a, b and similar)
  - Constants (1, 50, 300 and so on)
- It can also have a call to a function

```
#Python expressions
# Example 1
print(1+1)
Output:
2

# Example 2. The len() is a built-in Python function that returns the number of characters in a string
print(len("hello"))
Output:
5

# Example 3
y=3.14
print(y)
Output:
3.14

# Example 4
x=len("Python")
print(x)
Output:
6
```

Data Types

- **Data types** defines the type of the variable
- Dynamic Typing
- Get the data type of any object using the **type()** function
- Types
  - ❑ Numeric (int, float, complex)
  - ❑ Text (str)
  - ❑ Sequence (list, tuple, range)
  - ❑ Mapping (dict)
  - ❑ Boolean (bool)
  - ❑ Set (set, frozenset)
  - ❑ Binary (bytes, bytearray, memoryview)

Examples of Numeric Data Types

```
#int data type
num=5
print(num)
print ("Data type of variable num is", type(num))
Output:
5
Data type of variable num is <class 'int'>

#float data type
num1=3.14
print(num1)
print ("Data type of variable num1 is", type(num1))
Output:
3.14
Data type of variable num1 is <class 'float'>

#complex data type
Cnum=3+4j
print(Cnum)
print ("Data type of variable Cnum is", type(Cnum))
Output:
(3+4j)
Data type of variable Cnum is <class 'complex'>
```

Examples of Text Data Type

- String is a sequence of characters in Python
- The data type of String is called **str**
- Strings are either enclosed with **single quotes** or **double quotes**

```
# Python program to print Text data type such as string
s1 = "This is a String"
s2 = 'This is also a String'
# displaying string s1 and its type
print(s1)
print(type(s1))
# displaying string s2 and its type
print(s2)
print(type(s2))
Output:
This is a String
<class 'str'>
This is also a String
<class 'str'>
```

Examples of Boolean Data Type

- Boolean in Python can have two values – **True or False**
- When we compare two values, the expression is evaluated and Python returns the Boolean answer
- The **bool()** allows to evaluate any value

```
# Compare two values in an expression
print(10 > 9)
print(10 == 9)
print(10 < 9)
# Evaluate a string and a number
print(bool("Hello"))
print(bool(15))
# Evaluate two variables
x = "Hello"
y = 15
print(bool(x))
print(bool(y))
```

Output:

```
True
False
False
True
True
True
True
```

Operators

- Operators are used to perform operations on variables and values
- Categories
  - ❑ Arithmetic
  - ❑ Assignment
  - ❑ Comparison
  - ❑ Logical
  - ❑ Identity
  - ❑ Membership
  - ❑ Bitwise

- Description of some categories
  - Arithmetic
    - Used with numeric values to perform common mathematical operations
    - + - * / % ** //
  - Assignment
    - Used to assign values to variables
    - = += -= *= /= %= **= //= &= |= ^= >>= <<=
  - Comparison
    - Used to compare two values
    - == != > < >= <=
  - Logical
    - Used to combine conditional statements
    - and or not

Operators Precedence and Associativity

- Precedence
  - Order of evaluation
  - Python interpreter evaluates operators with higher precedence first
- Associativity
  - Operators at the same precedence level are checked for associativity
    - Left-Right
    - Right-Left

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+x, -x, ~x	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
	|	Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

Questions/Comments