TEMASEK
JUNIOR COLLEGE

Temasek Junior College
2022 JC1 H2 Computing
Practical 4 – Python Basic Data Types

Session Objectives
By the end of this session, you will learn:
(i) what string, integers, floats and booleans are.
(ii) the `type()` and `isinstance()` function to determine object type.
(iii) type conversion and type casting.

§4 Basic Data Types

Python consists of many data types. They can be basic data types such as:
- strings
- numbers
- Booleans

or collection data types such as:
- lists
- tuples
- dictionaries
- sets

In this practical session, we shall learn how to work with some basic data types in Python.

§4.1 Strings

A string is a sequence of characters.

You would have know by now that a string is enclosed within a pair of single quotation marks ' ' or a pair of double quotation marks " ".

Python supports multi-line entry of strings with the use of three pairs of quotation marks.

Exercise 1
Enter the following code into the REPL. Observe what happens when each part is executed. (Note: The prompt representation >>> need not be entered.)

(a)
```
>>> '''hello
world'''
```

(b)
```
>>> print('''hello
world''')
```

(c)
```
>>> """hello
world"""
```

(d)
```
>>> print("""hello
world""")
```

You would have also known by now that you can assign strings as values to variables. In fact you can make use of variables to print a statement.

Exercise 2
Enter the following code into the Python file editor and run the code. Observe what happens.

```
w = 'I'
x = 'am'
y = 'from'
z = 'Temasek JC'
print(w, x, y, z)
```

You can find the length of a string using the `len()` function.

For a string containing multiple words, the length of the string includes the spaces between the words.

Exercise 3
Enter the following code into the Python file editor and run the code. Observe the lengths of the strings displayed.

```
x = len('hello')
print(x)
y = len('world')
print(y)
z = len('hello world')
print(z)
```

§4.1.1 Use of Escape Characters in Strings

Similar to other programming language, Python uses \ as escape character in strings.

Exercise 4
Enter the following codes in the REPL to find out more about the use of escape characters.

(a) Use of tab spacing: `\t`

```
print('hello\tworld')
```

(b) Use of backslash: `\\`

```
print('Yes\No')
print("Yes\\No")
```

(c) Printing the text in a separate line or newline: `\n`

```
print('Hello\nworld')
```

Compare this with the use of three pairs of quotation marks in **Exercise 1**.

(d) Use of a single quotation marks in a string enclosed by a pair of single quotation marks: `\'`

```
print('I'm from Temasek JC.')
print('I\'m from Temasek JC.')
```

(e) Use of double quotation marks in a string enclosed by a pair double quotation marks: `\"`

```
print("He said "I am from Temasek JC".")
print("He said \"I am from Temasek JC\".")
```

Parts **(d)** and **(e)** in **Exercise 4** can be achieve using an alternating combination of single and double quotation marks instead of \ as escape character.

Exercise 5

Enter the following codes in the REPL to find out more about using single and double quotation marks in alternating combination.

(a) `print("He said 'I am from Temasek JC'.")`

(b) `print('He said "I am from Temasek JC".')`

(c) `print("I'm from Temasek JC.")`

In **(a)**, a pair double quotation marks was used to enclose the string and a pair of single quotation marks was used to enclose the speech. In **(b)**, the use of the double and single quotation marks was reversed.

Both the methods in **(a)** and **(b)** will work for **(c)**. However, it is more natural to use the single quotation mark as an apostrophe.

§4.1.2 Using the Addition + and Multiplication * Operators on Strings

Exercise 6

Enter and run the following statements on REPL. Observe what happens.

(a) `'a' + 'a'`

(b) `'abc' + 'xyz'`

(c) `'a' * 3`

(d) `'xyz' * 4`

(e) `'a' * 'a'`

(f) `'abc' * 'xyz'`

(g) `'a' * 2.5`

(h) `'a' * -1`

Notice that strings can be added together as seen in **(a)** and **(b)**. We call this string **concatenation**.

The same string can also be concatenated multiple times by multiplying it to the number of times it needs to be concatenated as seen in **(c)** and **(d)**.

However, two strings cannot be multiplied together as seen in **(e)** and **(f)**.

Strings also cannot be multiplied to a non-positive integer as seen in **(g)** and **(h)**.

§4.2 Numbers

Numbers in Python can be classified into integers, float and complex.

Integers are whole numbers (positive, negative and zero). They are classified as `int` in Python.

Floats are decimal values (non-integer real numbers). They are classified as `float` in Python.

Complex numbers are numbers of the form `m + nj` or `m + nJ` when `m` and `n` are real numbers and `j` or `J` is written behind `n` to make the entire value complex.

The type function `type()` and the `isintance()` functions can be used to the data type. Let us see how they can be used.

Exercise 7
Enter and execute the following code in the REPL. Observe the results.

(a)	`type(123)`	**(e)**	`isinstance(123, int)`
(b)	`type(12.3)`	**(f)**	`isinstance(123, float)`
(c)	`type(1.2 + 3j)`	**(g)**	`isinstance(12.3, int)`
(d)	`type(1.2 + 3J)`	**(h)**	`isinstance(12.3, float)`

Question
What do you think is the output of the following codes? Why?

```
type('123')
isinstance('123', str)
```

Answer

<class 'str'>
True.

§4.2.1 Use of _ as Spacing between Digits

We often observe that long integer values are separated with a spacing for ease of reading e.g. the value 1000000 is usually written as 1 000 000.

In Python, the _ is used instead of a spacing.

Exercise 8
Enter and execute the following code in the REPL. Observe what happens.

```
x = 1 000 000
x = 1_000_000
print(x)
```

§4.2.2 Mathematical Operators

The following operators are used to perform arithmetic operations

- Addition `+`
- Subtraction `-`
- Multiplication `*`
- Division `/` (gives the actual value of division as `float` type)
- Floor Division `//` (gives the quotient of division as `int` type)
- Modulo `%` (gives the remainder of division)
- Exponentiation (Power) `**`

Exercise 9
Evaluate the following expressions on the REPL and write down the answers.

(a) `8 + 3`
(b) `8 - 3`
(c) `8 * 3`
(d) `8 / 3`
(e) `8 // 3`
(f) `8 % 3`
(g) `8 ** 3`

A division operation by the / operator will always result in a `float` data type even if the division is exact. To obtain an `int` data type in this circumstance, the floor division `//` operator should be used.

Exercise 10
Enter and run the following statements on REPL. Observe what happens.

```
x = 6 / 3
type(x)
y = 6 // 3
type(y)
```

§4.3 Booleans

Boolean data consists of only 2 values:

- `True`
- `False`

It is important that the first character T or F be used in the **upper case** when writing Boolean data

This data type is named after the Mathematician George Boole, the creator of Boolean algebra. Boolean logic laid the foundations for the information age.

We have seen in **Exercise 7** how the `isinstance()` function returns a Boolean result. We can reaffirm this by using the `type()` function.

Exercise 11
Enter and execute the following code in the REPL. Observe what happens.

```
type(True)
type(true)
type(False)
type(false)
```

Boolean data is commonly associated with the use of comparison operators. We shall learn more about the use of comparison operators in the next practical session.

§4.4 Type Conversion and Type Casting

There may be times where you want to convert a variable from one type to another. This can be done type casting with type constructor functions. Let us look at some type constructor functions based on what we know thus far:

- `int()` – constructs an integer number
- `float()` – constructs a non-integer real number
- `str()` constructs a string

Exercise 12

Enter and execute the following code in the REPL. Observe what happens:

```
x = '123'
type(x)
y = int(x)
type(y)
type(x)
```

The variable `x` is assigned the string '`123`'. Hence `x` is of data type string. The variable `y` is assigned the integer of `123` by using `int()` to convert the string value of `x` to integer value. Hence `y` is of data type integer. However the value of `x` remains as a string.

Exercise 13

Enter and execute the following code in the REPL. Observe what happens:

(a)
```
x = 123.45
y = int(x)
y
```

(b)
```
a = 123.75
b = int(a)
b
```

Notice that the decimal places are dropped off when the `int()` is used. The number is truncated, not rounded off.

Exercise 14

Enter and execute the following code in the REPL. Observe what happens:

(a)
```
x = 'abc45'
type(x)
y = int(x)
z = float(x)
```

(b)
```
a = 12345
type(a)
b = str(a)
type(b)
b
```

Notice that `int()` and `float()` works only when the variable contains completely digits. It does not work when there is a mixture of letters and digits.

However `str()` still works when the variable contains completely digits and no letters.