Python Fundamentals

conversions, parsing, and reading/writing with the console

Objectives:

- Reading from the console
- Writing to the console
- Implicit conversion
- Explicit conversion (casting)
- Parsing strings to numbers

Implicit conversion

- We’ve learnt about operators and data types. But what happens when we mix numeric data types? For example, we multiply an **int** value by a **float** value.

- When two numeric operands are of different types, the one with the "smaller" type will be converted to the "larger" type and the arithmetic will be of the "larger" type.

- This is known as an implicit conversion.

Implicit conversion

- Here are the numeric types listed from smallest to largest.

  int, float, complex

- So, in our previous scenario, the **int** would be implicitly converted into a **float**. Because float is “bigger” than int.

Complex numbers

- A quick note on **complex** numbers. A complex number is a number involving a *real* and *imaginary* component.

- For example, `39 + 3i` is a complex number where `39` is the *real* portion, and `3i` is the *imaginary* portion.

- In Python we write the imaginary portion with a `j` instead of an `i`. So the above would be `39 + 3j`. Python provides this type for complex mathematical equations.

Implicit conversion

```
integer_num = 2
float_num = 5.2

result_num = integer_num * float_num
```

In the above example, the multiplication would result in `integer_num` being converted to a **float**. The result of that expression would be a **float** stored in the variable `result_num`

Explicit conversion

- There may be times where you want to explicitly convert the result of an expression to a specific type.

- For example, you want to convert a **float** into an **int**. For these situations you need to do what's called an *explicit conversion*, or a *cast*.

- Python provides helper functions to do this, in the form of `int()` and `float()`.

Implicit conversion

```
integer_num = 2
float_num = 5.2

result_num = integer_num * int(float_num)
```

In the above example, the multiplication would result in `float_num` being converted into an **int**. The result of that expression would be an **int** stored in the variable `result_num`.

Explicit conversion

- An explicit conversion always carries a risk, however.
- In the case of converting a **float** to an **int**, the decimal portion of the number will be removed. This is often called a *loss of precision*.
- In addition, the conversion may fail because Python can't successfully convert one value to another. So use it only when you know what you're doing.

Basic input-output

- One of the most fundamental parts of any programming language, is the ability to capture user input and display output.
- Python is no different, and provides simple ways to display information.
- Specifically, there are the `print()` and `input()` functions.

Basic input-output

- In simple scenarios this information will be displayed to the console, or captured from the console.

- What's the console? Well it depends. But some examples of console windows are:
  - Terminal on macOS
  - Terminal on Linux
  - Command Prompt on Windows
  - Windows PowerShell

print function

The print function takes a value and, by default, writes it to the console. The value could, for example, be a **string**, **integer**, or **float**. Below are some examples.

```
print("Hello World!")
# Printed value: Hello World!

print(20)
# Printed value: 20

print(3.14159)
# Printed value: 3.14159
```

print function

As well as taking a single value, the print function can take multiple values. Each value is called an *argument*, and is separated by a comma. Python will, by default, place an empty space (whitespace) between each value when displaying them.

```
print("This", "is", "a", "sentence.")
# Printed value: This is a sentence

print(1, 2, 3, 4, 5)
# Printed value: 1 2 3 4 5
```

print function

If you want to control what the print function places between the outputted arguments, you can use the sep keyword before an argument. This is called a *keyword argument*. Arguments without a keyword are called *positional arguments*.

```
print(2, 4, 6, 8, 10, sep=',')
# Printed value: 2,4,6,8,10

print("snake", "case", "naming", sep="_")
# Printed value: snake_case_naming
```

Special characters

A backslash character (\) in a string signifies a character escape sequence to allow the representation of non displayable characters. The most common sequences are shown below.

escape sequence	*character*
\n	newline
\t	tab
\"	double quote
\'	single quote
\\	backslash

Special characters

```
str = "Tom said,\n\n\t\"Hi!\""
print(str)
```

```
Tom said,

        "Hi!"
```

print function

You'll notice that the print function writes to a new line every time it's called. By default Python will place a new line character at the end of every function call. This can be altered by using the *keyword argument* `end`.

```
print("Hello", end="\t")

print("World!")

# Printed value after two function calls: Hello   World!
```

input function

As mentioned before, Python also has a way to capture user input from the console. For this we use the input function. It takes an optional argument, to prompt the user. Below is an example.

```
typed_text = input("Please enter a number between 1 and 10: ")
# Console displays: Please enter a number between 1 and 10:
# Waits for user to input a value and press enter
# Store the user input in a variable

# Print the variable value
print("Your number is...", typed_text)
```

input function

Note that the input function will always return a string value. This poses a problem if we want to do a calculation with the user input. Fortunately, explicit conversion helps us here as well.

```
typed_text = input("Please enter a number between 1 and 10: ")
number = int(typed_text)
# Convert the string input into an integer

print("Your number multiplied by itself is", number * number)
# Would print out 81 if the number was 9
```

Inputting data

- A quick note about converting data provided by users into another data type. This is prone to errors, and if not handled correctly will cause your script to crash.

- For example, let's assume the user typed in the value **five** when prompted for a number. Attempting to convert this into an integer would cause an error in our code.

- Validating user input is essential if you plan to develop any serious script or application.

References

Python documentation for print

https://docs.python.org/library/functions.html#print

Python documentation for input

https://docs.python.org/library/functions.html#input

Demonstration:

- Reading from the console
- Writing to the console
- Implicit conversion
- Explicit conversion (casting)
- Parsing strings to numbers