Python Workshop

1. Introduction

Quick jump: Hello, world | I/O | Advanced topics

Workshop Objective

- Understand the basis of programming.
- Learn how to program with Python.
- Be able to learn other programming languages by yourselves.

Plan (Day 1)

Section	Topic
1	Introduction, language syntax, variables, basic I/O
2	Control structures

Plan (Day 2)

Section	Topic
3	Data Structures
4	List opertions
5	String operations

Plan (Day 3)

Section	Topic
6	Functions
7	Classes
8	Futher references

Format

- This workshop runs in a format that encourage self-learning:
  - Quick introduction of topics (5-15 min)
  - Self-learning materials and exercises (30-60 min)
  - Demonstrations (15-30 min)
- Pacing will be adjusted based on your progress.

Why self-learning

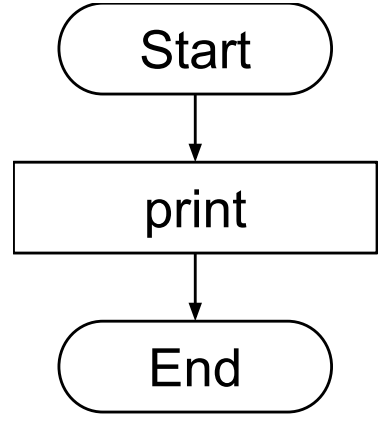
- The best way to learn programming is by practicing.
- Everyone has their own pace in learning.
- You are encouraged to ask more questions than just listening.
- We can answer your questions individually.

Hello, world

- We recommend the use of **Spyder** for this workshop.
- Spyder comes with the installation of Anaconda, a distribution of programming language Python and R.
- To start Spyder in the lab: Search "Spyder" in the start-menu.

What is a program?

- A program usually defines a **sequence** of statements to be executed one by one.
- In a sequential, synchonous program, a program defines a flow of control that perform a task.
    - We can always draw a flow chart to represent such a program.



Hello, world

- Here is our first program. We can write this in Spyder, save it and run it.

```
# Hello, world
print('Hello, world!')
```

- Remember to save your work in your own device, files in the machines in the lab will be gone when you log out.

Hello, world explained (1)

- There are only two lines in our first program:
- The first line is a comment, it documents **what** we are doing in the program.

```
# Hello, world
```

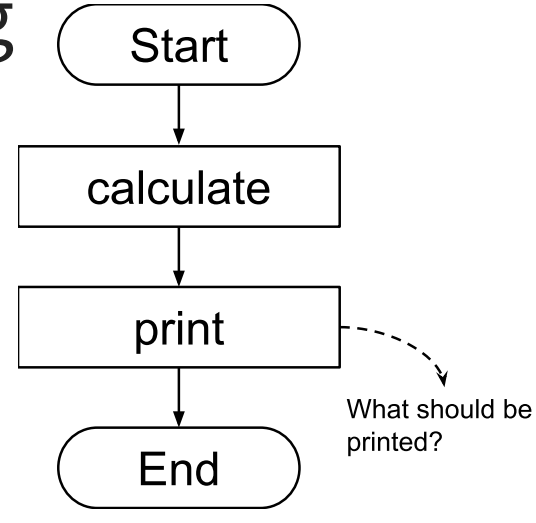
Hello, world explained (2)

- The second line is an **output** statement, it print the string "Hello, world!" to the console.

```
print('Hello, world!')
```

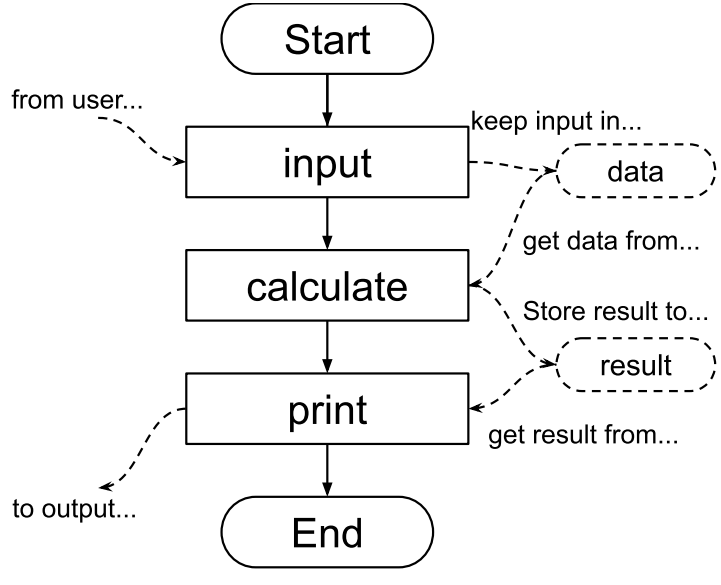
Concepts: Program state

- To allow steps in a program working together, we need to maintain the program state.
- We can use **variables** to maintain such state.
- Program can accpet input to alter the state.
- Program can output information according to the state.



Concepts: variables and I/O

- Variables: maintain program state
- Input: collect information from user/external
- Output: provide information to user/external



Variables, input and output

- The next example highlights the topics to be discussed in this section:

```
mystr = input('Please input your name:')
print('Hello', mystr)
```

- **`input()`** is used to ask user for an input, in the form of a string.
- **`=`** operator is used to assign the result of `input()` to a variable named `mystr`.
- **`print()`** is used to print some text.

Self-learning topics (~45min)

- Use of variables and printing of values
- Arithmetic operators
- Reading input from user
- Advanced topics:
  - Multiline string
  - Bitwise operation
- You are encouraged to copy-and-paste the code to Spyder and test them.

Variables

Variables

- To maintain program states, we associate values to names.
- These names are called variables.
- In Python, we use the **assignment operator** `=` to assign a value to a variable, for example:

```
a = 10
```

Values and Variables

```
a = 10
```

- In many programming language, variable must be **declared** before use.
    - In Python, assigning a value to a name automatically declare the variable.
- The **assignment operator** `=` will always assign the **value** on the right to the **variable** on the left.
- In the above example, variable `a` is assigned a value of 10.

Using variables

- If we assign a new value to a variable, the variable will be overwritten. For example:

```
a = 10
a = 20
```

- In the above example, variable `a` is assigned a value of 20.

Types of values

- A variable can be used to hold different types of values.
- Typical variable type in Python are **boolean**, **integer**, **floating-point**, and **string**.

```
# boolean values
a = True
b = False
# integer/floating-point value
c = 3
d = 1.5
# string (same for single quote or double quote)
e = "hello"
f = 'world'
```

Basic output

Printing multiple values (1)

- It is common to print a combination of values and variables. Consider this program for example:

```
a = 1
b = 2
c = 3
print(a)
print('+')
print(b)
print('=')
print(c)
```

- The output is not desirable (try it!), we pefer printing them on the same line.

Printing multiple values (2)

- We can print a list of values, separated by **commas** when we use `print()`, a space will be added between the values:

```
a = 1
b = 2
c = 3
print(a, '+', b, '=', c)
```

The output will be:

```
1 + 2 = 3
```

`sep`

- To separate the values to be printed by other string, we can specify the **`sep`** option in `print()`.

```
print(1, 2, 3)
print(1, 2, 3, sep=',')
```

Output:

```
1 2 3
1,2,3
```

end (1)

- Another option for `print()` is **end**, which control how to end the printing.
- By default a new line is inserted.

```
print(1, end=' + ')
print(2, end=' = ')
print(3)
```

Output:

```
1 + 2 = 3
```

end (2)

- We can even use an empty string:

```
print(1, end='')
print(2, end='')
print(3)
```

- Output:

```
123
```

- Note that we keep the last print as default to insert a new line at the end.

Arithmetic operators

Arithmetic operators (1)

- For integer and floating point numbers, we can use the four **arithmetic operators**: `+`, `-`, `*` (multiply), and `/` (divide). For example:

```
print(1 + 2)      # output 3
print(10 - 1.5)   # output 8.5
print(2 * 6)      # output 12
print(9 / 6)      # output 1.5
```

Arithmetic operators (2)

- Order of execution follows basic mathematics rules.
- For example:

```
print(1 + 2 * (3 - 4) / 5)  # output 0.6
```

Floating point division

- Operator `/` always results in a **floating point number**, even when the operands are integers.

```
a = 100
b = 10
print(a / b)  # output 10.0
```

Floor division

- If **integer division** is needed, we can use the `//` operator instead. This operator will perform division and return the floor of the result. For example:

```
print(9 // 6)      # output 1
print(100 // 10)   # output 10
```

Power operator

- The **power operator** `**` calculate and return the value of a base raised to a specific power.

```
print(10 ** 2)  # output 100
print(2 ** 10)  # output 1024
```

- Remeber: do not use the `^` operator, which is another operator (discussed in advanced topics).

Modulo operators

- The **modulo** operator % calculates and returns the **remainder** of dividing first operand by the second operand.

```
print(100 % 3)  # output 1
print(100 % 7)  # output 2
```

- **This operator is extremely important in computer science.** (why?)

Reading input

Using `input()`

- The `input()` function will always read a string from user.

```
a = input('Please input a string:')
print(a)
```

- The part 'Please input a string:' is a message to prompt user for input. It can be omitted:

```
a = input()
print(a)
```

Reading integer or floating-point values

- `input()` will always return a string, before we can use an input value in arithmetic calculation, we need to convert it:
- For example:

```
a = int(input())
b = float(input())
print(a, '+', b, '=', a + b)
```

Exercises

- Exercises help you to check your understanding to the topics.
- There is no specific requirement to these exercises, you are encouraged to explore different possibilities.

Exercise 1 (1)

- Write a program that reads **two floating-point values** and compute their hamonic mean.
- Harmonic mean H of input values a, and b can be calculated by the formula $\frac{1}{H} = \frac{1}{2}(\frac{1}{a} + \frac{1}{b})$, or simply $H = \frac{2ab}{a+b}$.

Exercise 1 (2)

- Sample input/output as follow:

Input	Output
1, 4	1.6
3, 7	4.2
7, 9	7.875
3.7, 4.3	3.9775

Exercise 2

- Write a program that convert time period (in seconds) to the long format represented by the pattern `?h ?m ?s`.
- Sample input/output as follow:

Input	Output
100	0h 1m 40s
10000	2h 46m 40s
1000000	277h 46m 40s

Advanced topics

- At the end of each section, we will include some concepts that is comparably less essential but worth to know.
- You may decide to skip them and move on to the next section.

Multiline string

Multiline string

- Multi-line string is defined using three single quotes or double quotes:

```
a = '''Hello,
world!'''
b = """Hello,
world!"""
```

Multiline comments

- Often the multi-line string notation is used to represents multi-line comments:

```
'''
This is a comment.
This string is not kept in any variable.
'''
```

- Multiline comments is also used for documentation of codes.

Number base and Bitwise operators

Number base

- Apart from base 10 numbers, we can define numbers with base 2, 8 and 16 in Python.
  - Base 2 number is prefixed by pattern `0b`.
  - Base 8 number is prefixed by pattern `0o`.
  - Base 16 number is prefixed by pattern `0x`.

```
print(0b011010100)   # output 212
print(0o324)         # output 212
print(0xD4)          # output 212
```

Binary representation

- In a computer, all values are stored as binary numbers.
- So number `212` is internally stored as `0b11010100`.
- These numbers are left-padded with zeros to match with the bit-length. Therefore the number `212` is actually stored as `0b00...011010100`.
- Note that in a signed representation, numbers are stored in 2's complement representation.

Bitwise operations

- Bitwise operations apply on numbers bit by bit, for example, the AND operation (`&`) on values 12 (`0b01100`) and 10 (`0b01010`) will be:

```
  01100 (12)
& 01010 (10)
------------
  01000 (8)
```

- Only one of the bits above will give a result of 1 as both operands are 1.

Bitwise Logical Operators (1)

- Bitwise logical operators includes AND (&), OR (|), and XOR (^), for example:

```
a = 0b01100
b = 0b01010
print(a & b)   # output 8
print(a | b)   # output 14
print(a ^ b)   # output 6
```

Bitwise Logical Operators (2)

- There is also the negation operator (`~`) which inverts all the bits. For example, positive value `01100` (`00...01100`) will becomes `11...10011`.
- In 2's complement representation. The above value equals `-13`.

```
a = 0b01100
b = 0b01010
print(~a)  # output -13
print(~b)  # output -11
```

Shift Operators

- Shift operators shift the binary pattern to the left (`<<`) or right (`>>`). For example:

```
x = 0b01101
print(x >> 2)   # output 3
print(x << 1)   # output 26
```

- `x >> 2` shifts value `0b01101` (13) two position to the right, therefore result is `0b00011` (3).