Imam Abdulrahman Bin Faisal University
Deanship of Preparatory Year and Supporting Studies

Basics of Programming through Python
Why should you learn to write programs?

Introduction to Programming COMP102

Term 3 2022-2023

Learning outcomes

- Discuss why we need to learn programming.
- Understand the basic building blocks of programs in Python.

Test your Knolwadge

- What is the difference between variables and constants?
- What do you know about data type in PL?
- What do you know about the input of any program?

Uunderstanding programming

Programming

- You need two skills to be a programmer:
- First, you need to know the programming language (Python) - you need to know the **vocabulary** and the **grammar**.
  - You need to be able to spell the words in this new language properly and know how to construct well-formed “sentences” in this new language.

Programming

- Second, you need to “**tell a story**”.
  - In writing a story, you combine words and sentences to convey an idea to the reader.
- There is a ***skill*** and ***art*** in constructing the story, and skill in story writing is improved by doing some writing and getting some feedback.
  - In programming, our program is the “**story**” and the problem you are trying to solve is the “**idea**”.

Programming

- Once you learn one programming language such as Python, you will find it much easier to learn a second programming language such as JavaScript or C++.
- The new programming language has very different vocabulary and grammar, but the **problem-solving skills** will be the same across all programming languages.

Terminology: Interpreter and compiler

Terminology

- Python is a **high-level language** intended to be relatively straightforward for humans to read and write and for computers to read and process.
- The actual hardware inside the Central Processing Unit (CPU) **does not understand** any of these high-level languages.
- Instead, we build various **translators** to allow programmers to write in high-level languages like Python or JavaScript and these translators convert the programs to machine language for actual execution by

Terminology

- These programming language translators fall into two general categories:(1) **interpreters** and (2) **compilers**.
- An *interpreter* reads the source code of the program as written by the programmer, parses the source code, and interprets the instructions on the fly.
  - Python is an interpreter and when we are running Python interactively, we can type a line of Python (a sentence) and Python processes it immediately and is ready for us to type another line of Python.

Terminology

- A compiler needs to be handed the entire program in a file, and then it runs a process to translate the high-level source code into machine language and then the compiler puts the resulting machine language into a file for later execution.
- It is not easy to read or write machine language, so it is nice that we have ***interpreters*** and ***compilers*** that allow us to write in high-level languages like Python or C.

Writing a program

Python Program

- Typing commands into the **Python interpreter** is a great way to experiment with Python's features, but it is not recommended for solving more complex problems.
- When we want to write a program, we use a **text editor** to write the Python instructions into a file, which is called a **script**.
  - By convention, Python scripts have names that end with ***.py***.

The building blocks of programs

- There are some low-level conceptual patterns that we use to construct programs.
- These **constructs** are not just for Python programs, they are part of every programming language from machine language up to the high-level languages.
  - Input
  - Output
  - Sequential execution
  - Conditional execution
  - Repeated execution
  - Reuse

The building blocks of programs

- **Input:**
  - Get data from the “outside world”.
  - This might be reading data from a file, or even some kind of sensor like a microphone or GPS.
  - In our initial programs, our input will come from the user typing data on the keyboard.

The building blocks of programs

- **Output:**
  - Display the results of the program on a screen or store them in a file or perhaps write them to a device like a speaker to play music or speak text.
- **Sequential execution:**
  - Perform statements one after another in the order they are encountered in the script.

The building blocks of programs

- **Conditional execution:**
  - Check for certain conditions and then execute or skip a sequence of statements.
- **Repeated execution:**
  - Perform some set of statements repeatedly, usually with some variation.
- **Reuse:**
  - Write a set of instructions once and give them a name and then reuse those instructions as needed throughout your program.

What could possibly go wrong?

Errors / Exceptions

- As your programs become increasingly sophisticated, you will encounter three general types of errors:
  - Syntax errors
  - Logic errors
  - Semantic errors

Errors / Exceptions

- **Syntax errors**:
  - These are the first errors you will make and the easiest to fix.
  - A syntax error means that you have violated the “grammar” rules of Python.
  - Python does its best to point right at the line and character where it noticed it was confused.
  - The line and character that Python indicates in a syntax error may just be a starting point for your investigation.

Example

```
>>> primt 'Hello world!'
File "<stdin>", line 1
primt 'Hello world!'
SyntaxError: invalid syntax
>>> primt ('Hello world')
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'primt' is not defined

>>> I hate you Python!
File "<stdin>", line 1
I hate you Python!
SyntaxError: invalid syntax
>>> if you come out of there, I would teach you a lesson
File "<stdin>", line 1
if you come out of there, I would teach you a lesson
SyntaxError: invalid syntax
```

Errors / Exceptions

- **Logic errors:**
  - A logic error is when your program has good syntax but there is a mistake in the order of the statements or perhaps a mistake in how the statements relate to one another.
  - A good example of a logic error might be:
    - “take a drink from your water bottle, put it in your backpack, walk to the library, and then put the top back on the bottle.”

Example

```
x=3
y=2
print(z)
z=x+y
```

```
line 3, in <module>
    print(z)
NameError: name 'z' is not defined
```

Errors / Exceptions

- **Semantic errors:**
  - A semantic error is when your description of the steps to take is syntactically perfect and in the right order, but there is simply a mistake in the program.
  - The program is perfectly correct, but it does not do what you ***intended*** for it to do.

Example

```
x=3
y=2
z=x-y
print(z)
```

Operator in this example should be (+) but by mistake (-)

No Error Messgae displayes , and the program Run correctly

Errors / Exceptions

- When Python spits out an error or even when it gives you a result that is different from what you had intended, then begins the hunt for the cause of the error.
- ***Debugging*** is the process of finding the cause of the error in your code.

Thanks!

Any questions?