Introduction

Objective of this module (lesson):

- Cover some preliminaries:
    - Python as a programming language
    - Debugging
    - write our first Python program
    - analyze our first Python program
- learn about Python data:
    - Python data
    - Expressions and statements

Algorithms

An **algorithm**: a list of instructions that will solve a particular problem:

- Developing **algorithms** is a key goal in CS
- An **algorithm** presents the general solution
- We write it in a natural language like English
- We write a program to translate it to a language the computer can understand

Algorithms

Problem:

Write an algorithm to formulate a recipe for making a peanut butter and jelly sandwich. Steps:

1. Collect ingredients, bread, peanut butter, and jelly

2. Take one slice and spread peanut butter

3. Take the other slice and spread jelly

4. Put the two slices together and serve

Computer Program

Why do we need a **program**?:

- Computer only understand 0s and 1s; it doesn’t understand languages we speak
- Giving instructions in 0’s and 1’s is tedious!
- Also we solve problems in English; means we need a translator; so we can program in English
- We use programming languages to translate. Python: powerful and famous translator

Low-level languages

Two types of **low-level languages**:

- Machine: in binary format, 0s and 1s:
  - Computer can only run code in machine language
- Assembly: simple commands refer to instructions
- Advantages: runs fast; no or minimal translation
- Disadvantages:
  - Can run only on CPU the language designed for
  - Hard to read, too long and takes lot of time to write

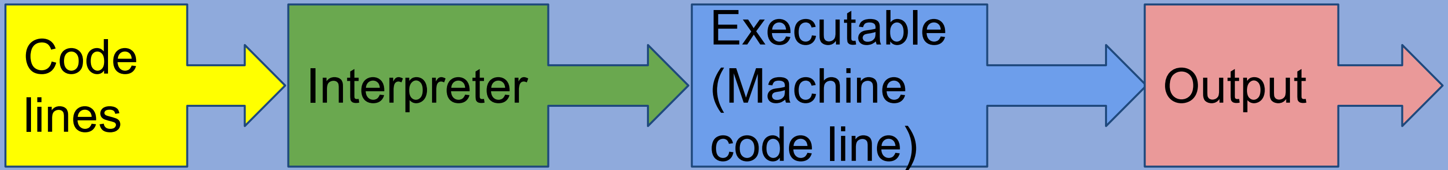
High-level languages

High-level languages mitigate difficulties of low-level:

- Advantages:
  - Written in English: easy to read
  - Short: less time to code
  - Less error prone
  - Portable: machine independent
- Need a translator to translate high-level code to low-level
- Disadvantages: Runs slow due to translation

Translators

Interpreters: translate source code into machine code line at a time and runs



Code lines

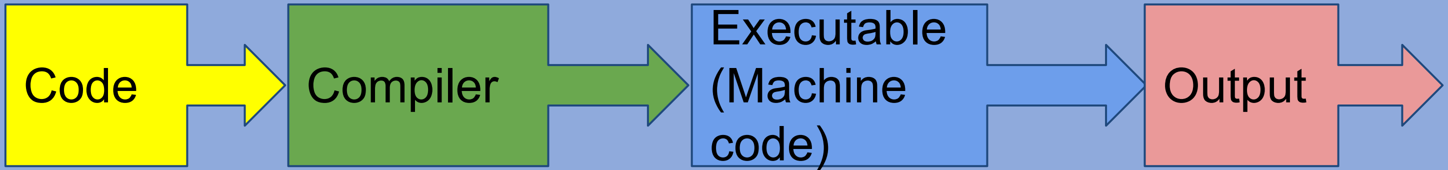
Interpreter

Executable (Machine code line)

Output

Compilers: Translate entire source code into machine code then runs entire machine code

- Can execute compiled code many times



Code

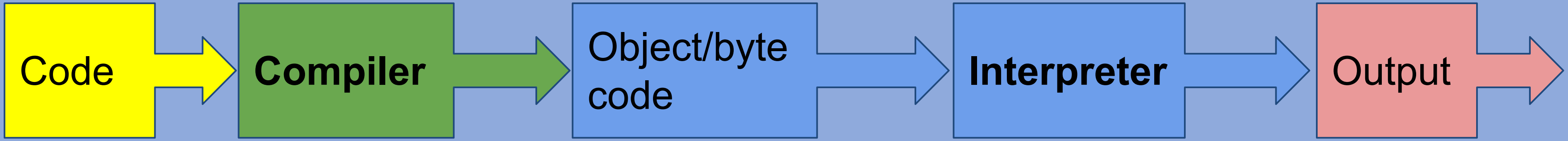
Compiler

Executable (Machine code)

Output

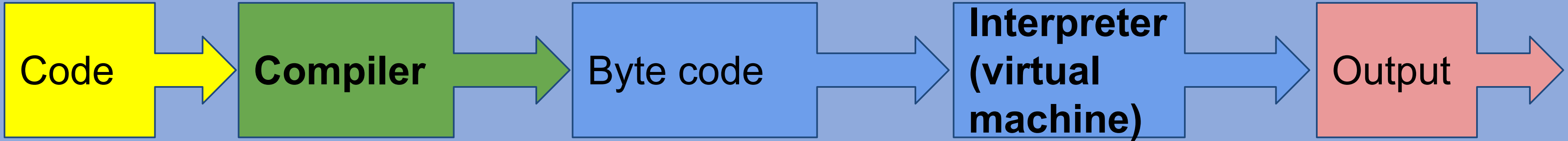
Python language

Some languages like Python use both translations:



Code → **Compiler** → Object/byte code → **Interpreter** → Output →

- Python :



Code → **Compiler** → Byte code → **Interpreter (virtual machine)** → Output →

- Two ways to run Python:
  - Shell mode: run simple code at >>>
  - Program mode: run source code saved in text files at >>>

Programming concepts

program: a sequence of instructions that specifies how to perform a computation

- It could be something simple or complex
- Specific syntax may be different between languages
- But all language share these features:
  - **Input**: get data from file or device(keyboard, etc)
  - **Output**: display data in screen or save data
  - **Math and Logic**: do basic math and logical operations
  - **Selection**: conditional execution
  - **Iteration**: repetition of tasks

Bugs and debugging

Errors in a program are call bugs:

- Debugging: Process of removing them
- Three main types of bugs:
  - **Syntax errors**: due to mistakes in grammar
  - **Logical or semantics errors**:
    - runs but not giving expected results
    - no errors return
    - logic and/or algorithm could be wrong
  - **Runtime errors**: only appear at runtime

The first Python program

Python

```
# my first program
# prints "Hello! World"
to the screen
print("Hello! World")
```

The first Python program

Indika walimuni PhD

Java

```
/**
 *  my first program
 *  prints the "Hello! World" in
the screen
 */
package hello;
public class Hello {
     public static void
main(String[] args) {
          System.out.println("Hello
! World");
     }
}
```

Python

```
# my first program
# prints "Hello! World"
in the screen
def main():
   print("Hello! World")


if __name__ == "__main__":
   main()
```

C++

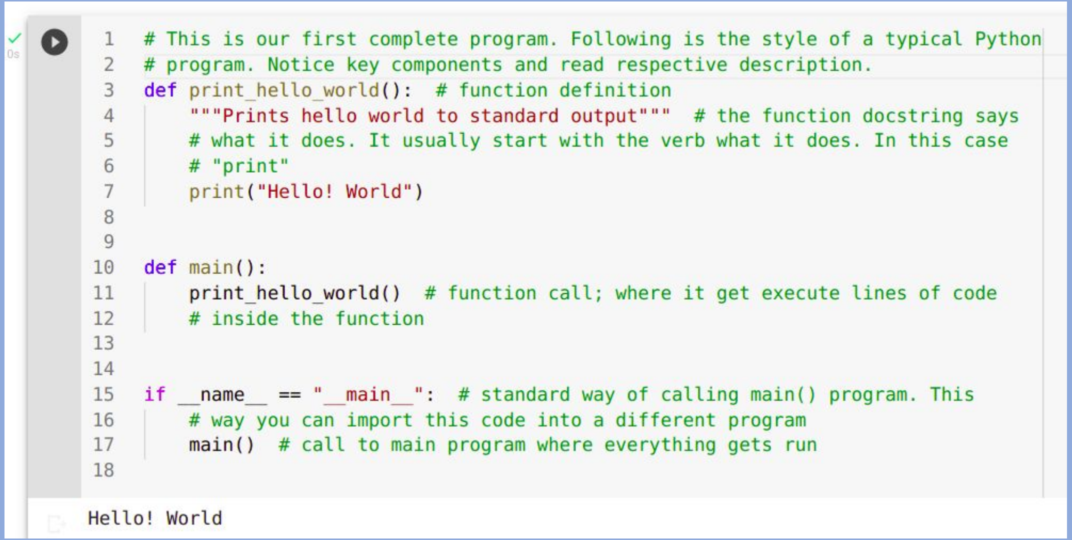
```
#include <iostream>
/**
 *  my first program
 *  prints "Hello! World" in
the screen
 */
int main(int argc, char
*argv[])
{
    std::cout << "Hello!
World" << std::endl;
    return 0;
}
```

The first Python program

Indika walimuni PhD

Let us analyse this program:

- *comments*: start with #. Python don’t run them
- *Function definition:* starts with `def` keyword
- *Docstring*: enclosed by """ """ or ''' '''special comment used for documentation
- `print()`: prints things we pass to it



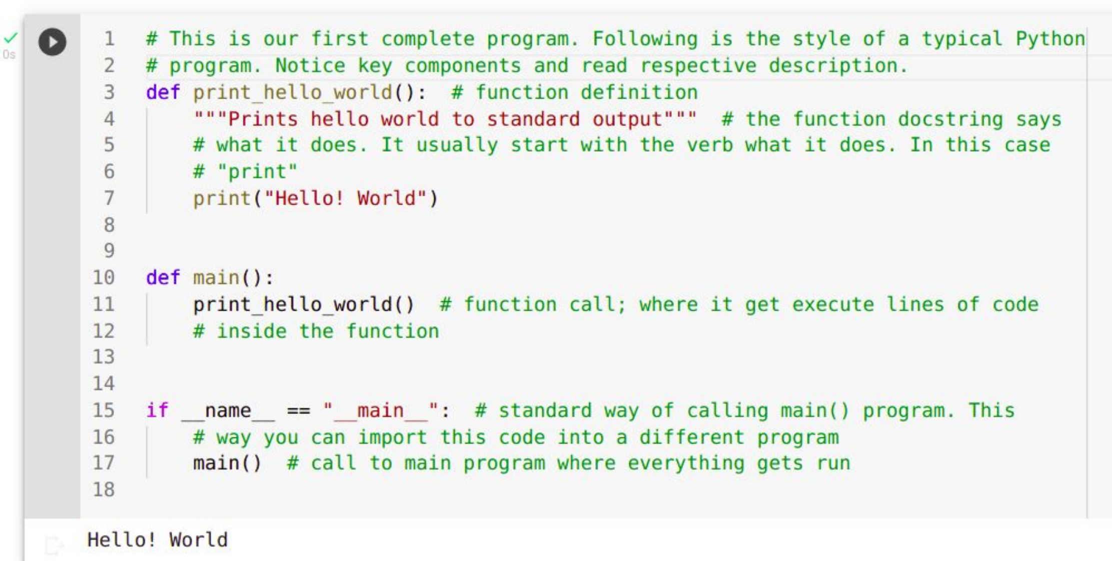
```
# This is our first complete program. Following is the style of a typical Python
# program. Notice key components and read respective description.
def print_hello_world():  # function definition
    """Prints hello world to standard output"""  # the function docstring says
    # what it does. It usually start with the verb what it does. In this case
    # "print"
    print("Hello! World")


def main():
    print_hello_world()  # function call; where it get execute lines of code
    # inside the function


if __name__ == "__main__":  # standard way of calling main() program. This
    # way you can import this code into a different program
    main()  # call to main program where everything gets run

```

```
Hello! World
```

The first Python program

Let us analyse this program:

- `def main()`: main program definition
- `if`: another keyword used in selection
- `main()`: call to main program



```python
# This is our first complete program. Following is the style of a typical Python
# program. Notice key components and read respective description.
def print_hello_world():  # function definition
    """Prints hello world to standard output"""  # the function docstring says
    # what it does. It usually start with the verb what it does. In this case
    # "print"
    print("Hello! World")


def main():
    print_hello_world()  # function call; where it get execute lines of code
    # inside the function


if __name__ == "__main__":  # standard way of calling main() program. This
    # way you can import this code into a different program
    main()  # call to main program where everything gets run

```

```
Hello! World
```

Values

A **value** is an entity that a program manipulates:

- Ex. we manipulated "Hello! World" as a value
- All values are Python objects
- Every object has a data type or a class, a value and an identity
- Use `type(a_value)` to figure type of `a_value`

Type

Every value has a data **type** or a **class** in Python:

- Ex. 20 is int (integer) and "Hello! World" is str (string)
- Type realized at runtime: Python is dynamically typed
- If semantics agree, may convert between types
  - Ex. `int("20")` → `20`
- Type intermingling not allowed: strongly typed
  - Ex. `'x' + 5` → `TypeError`
  - Above is allowed in JS: JS is weakly typed

Variables

Indika walimuni PhD

A **variable** is a name that refers to an object:

- Use variables for bookkeeping values
- Use = and assignment statement to create variables
  - Ex. `planet_name = "Mars"`
- In assignment statements, = only works left to right
  - Please don't follow algebra here
- Read it like, `planet_name` assigned to or referring to `"Mars"` or use the reference diagram,

  `planet_name` → `"Mars"`

Variables

Commonly names in Python called identifiers. There is a naming convention for naming identifiers:

- An identifier name:
  - must begin with a letter or an underscore
  - can’t have special characters or spaces
  - can’t use Python keywords
- Naming violations returns SyntaxErrors

- Always use expressive names

Identity

Every object has an **identity** in Python:

- **Identity**, value and type uniquely identify an object
- `id(an_object)` → the identity, id of `an_object`
- id value is a unique integer value related to the memory location of the value of the object
- Two different variables may have same id:
  - We say they are aliased
  - We will learn more on id later

Statements

Statement: a single instruction Python can run:

- Can be single line or multiline
- Assignment is a statement Ex, `course_umber = 20`
- Loops (`for, while`) selection (`if, else`) and `with` are called compound statements
- Often assignments silently get executed

Expressions

Expression: combination of values, operators, variables, function calls are expressions:

- Usually sits at the right hand side of assignment
- Evaluation of an expression returns a value
- Value or a variable itself is an expression
- Expressions can be parts of a statement