MONASH University

MONASH
INFORMATION
TECHNOLOGY

FIT9133 Semester 2 2018
Programming Foundations in Python

Week 1:
Introduction to Programming and Algorithms
Python Basic Data Types

Chunyang Chen

GROUP OF EIGHT AUSTRALIA

Module 1 Synopsis

- This module is aimed to provide you with:
  - Overview of the Python programming language
  - Interactive IDE for Python
    - Jupyter notebook (a.k.a. iPython notebook)
  - Basic concepts of programming
    - Programs and algorithms, notion of abstraction
  - Fundamental programming constructs in Python
    - Primitive data types, variables, expressions, statements, assignments, arithmetic and logical operators, standard input and output

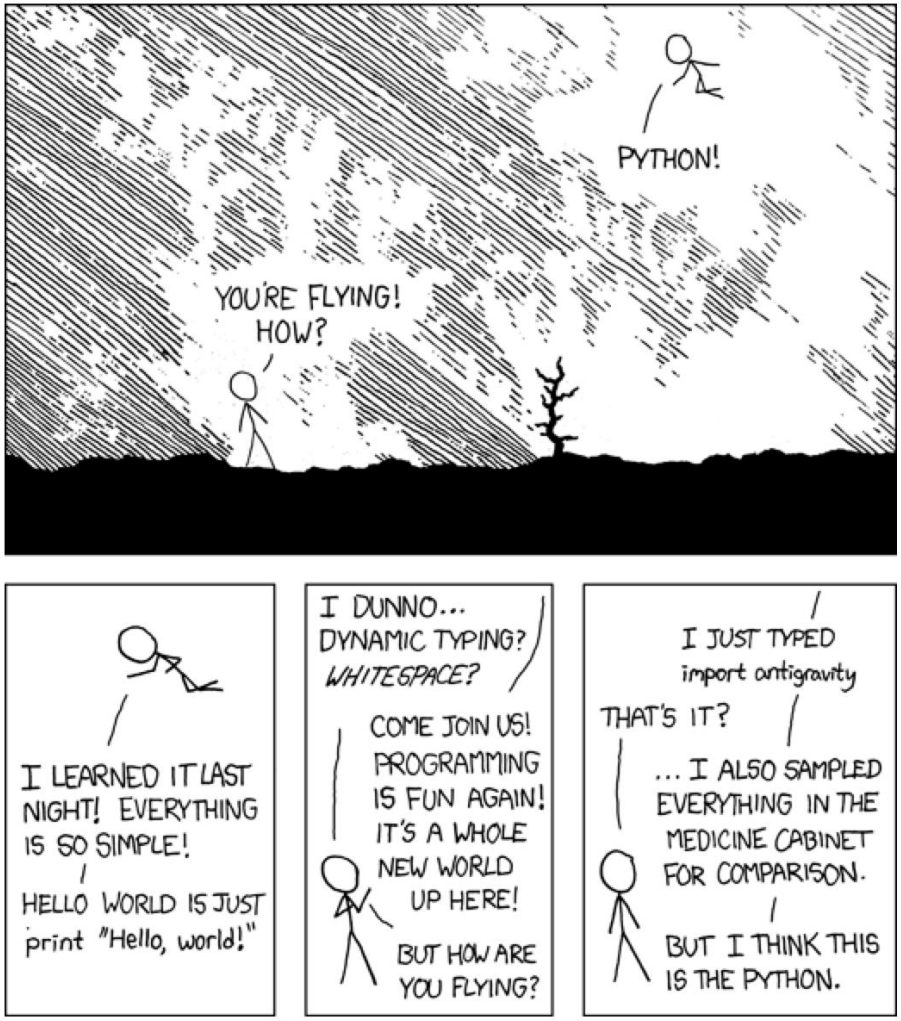
Module 1 Learning Objectives

- After working your way through this module, you should be able to:
  - Recognise a computational problem
  - Define an algorithm for solving the problem
  - Identify and use various programming constructs used in Python
  - Build basic computational programs in Python

MONASH
University

Introduction to Python

xkcd: Python



https://www.explainxkcd.com/wiki/index.php/353:_Python

Why Python?

- Python is a general-purpose programming language that can be used to literally develop any kind of programs.
- Python is a *simple* programming language that is easy to learn and use.
- Python is supported with a rich collection of libraries or packages (i.e. *ready-to-use* code) to build sophisticated programs.

"Lift is short
(You need Python)"

--Bruce Eckel
ANSI C++ Committee member

What Can Python Do?

- Scripting
  - Crawling
  - Calculation
- Website development
- Visualization
- Data Science
  - Computer Vision
  - Natural Language Processing
  - Speech Recognition
- Desktop application with GUI

Who Uses Python? [Chapter 1, Learning Python by Mark Lutz (2013)]

The Evolution of Python

- Invented in Christmas of 1989 by a Dutch programmer
  - Guido van Rossum
  - "Python" is named after Guido's favourite British comedy series "Monty Python's Flying Circus"

- Python 2 vs. Python 3:
  - Python 3 was released to address various design decisions and inconsistency in Python 2 (and its subsequent releases 2.x)
  - Python 3 is backward incompatible with Python 2.x
  - Note: We will use Python 3.7 (or Python 3.6/3.5) for this unit

High-level Programming Languages

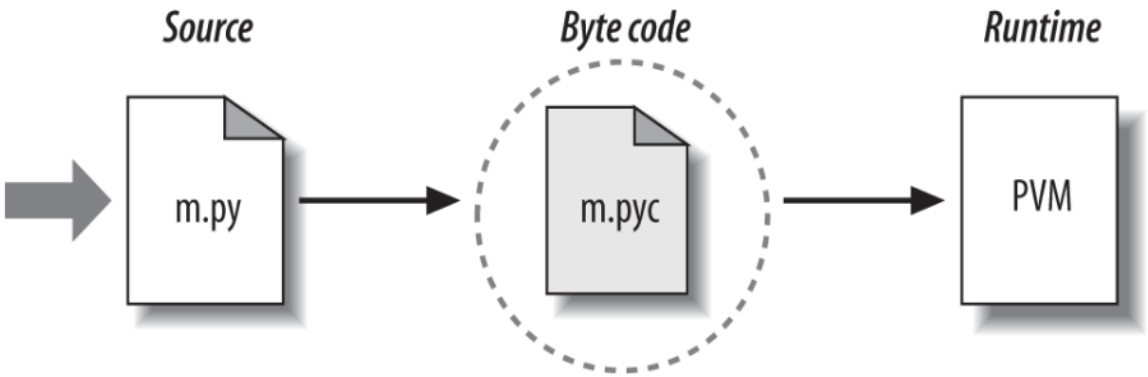
- What is a high-level programming language?
  - Not interpretable or directly executable by the CPU
  - Has to be translated into a *low-level* language (assembly language) or the executable machine language
  - E.g.: Python (C, C++, Java, Ruby, …)

- Advantages (as compared to low-level languages):
  - Easier to implement and require shorter amount of time to write
    - (Very) English like
    - More readable/comprehensible by programmers
  - Portable and platform independent
    - Programs just have to be written once but can be run on any types of computer platform (without much modification effort)

The Execution of Python Programs

- When you execute a Python program:
  - Source code of your program (`.py` files) is *compiled* (translated) into a format known as byte code
    - Byte code is a lower-level, platform-independent representation of the source code of your program
    - Byte code of your programs are stored as `.pyc` files in the subdirectory named __pycache__
- Once a program has been compiled into byte code (and whenever the byte code is loaded from the `.pyc` file):
  - Each of the byte code instructions will be *interpreted* and executed by the runtime execution engine, Python Virtual Machine (PVM)

Python's Runtime Execution Model



[Source: Chapter 2, Learning Python by Mark Lutz (2013)]

Running Python Programs

How to Execute Python Programs?

- Python programs can be executed by using a Python interpreter in two modes:
  - Interactive mode
  - Script mode

- Interactive mode:
  - Start up the Python interpreter by running the command “python” at the prompt of a command-line terminal
  - Type the Python statements at the interactive mode prompt represented by >>>

```
Python 3.5.2 |Anaconda 2.5.0 (x86_64)| (default, Jul  2 2016, 17:52:12)
[GCC 4.2.1 Compatible Apple LLVM 4.2 (clang-425.0.28)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

How to Execute Python Programs? (continue)

- Script mode:
  - Create the Python source file (a.k.a. the *module* file) with the file extension of `.py` using any text editor
  - Pass the source file (e.g. `program.py`) as an argument to the "python" command:

```
python program.py
```

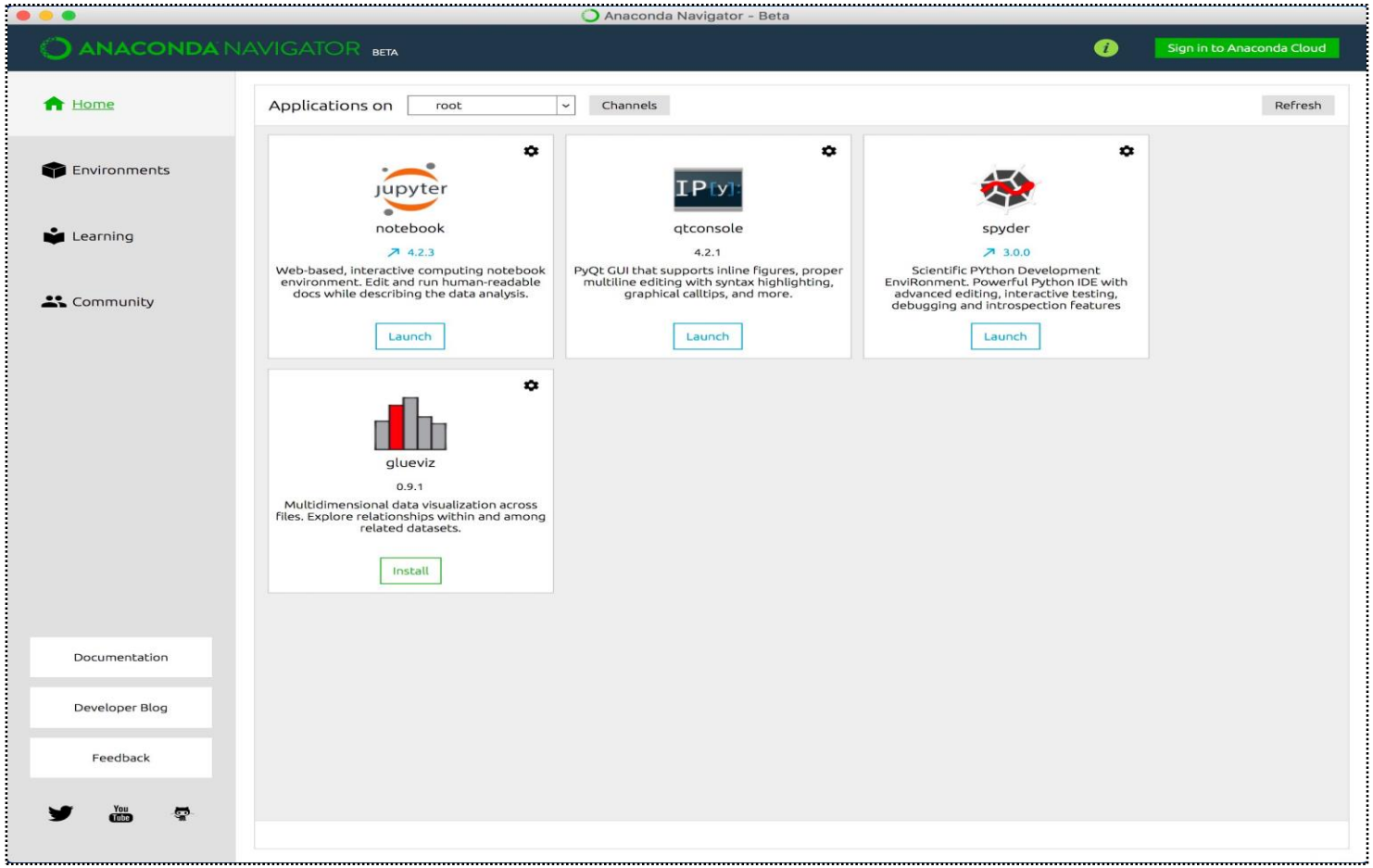
Python IDE: Anaconda (Jupyter Notebook)

- Integrated Development Environment (IDE):
  - More manageable and convenient to develop Python programs
  - Equipped with a set of utilities and libraries (packages)
  - Assist programmers to edit, build, and debug their programs
  - E.g. IDLE (the default Python IDE), PyCharm, Jupyter Notebook (an interactive web-based IDE)

- Anaconda:
  - Python distributions especially for data science.
  - Bundled with more useful packages for scientific computation and data analysis (e.g. NumPy, SciPy, Pandas, etc.)

Running Python with Jupyter Notebook

- `



[Anaconda Launcher Window]

Running Python with PyCharm

[PyCharm Launcher Window]

Programming Tools

- Stack Overflow
  - Largest Q&A site about programming
  - 17M questions, 26M answers, 10M users

- GitHub
  - Largest host of source code
  - 57M repositories including 28M public ones

Fundamental Concepts of Programming

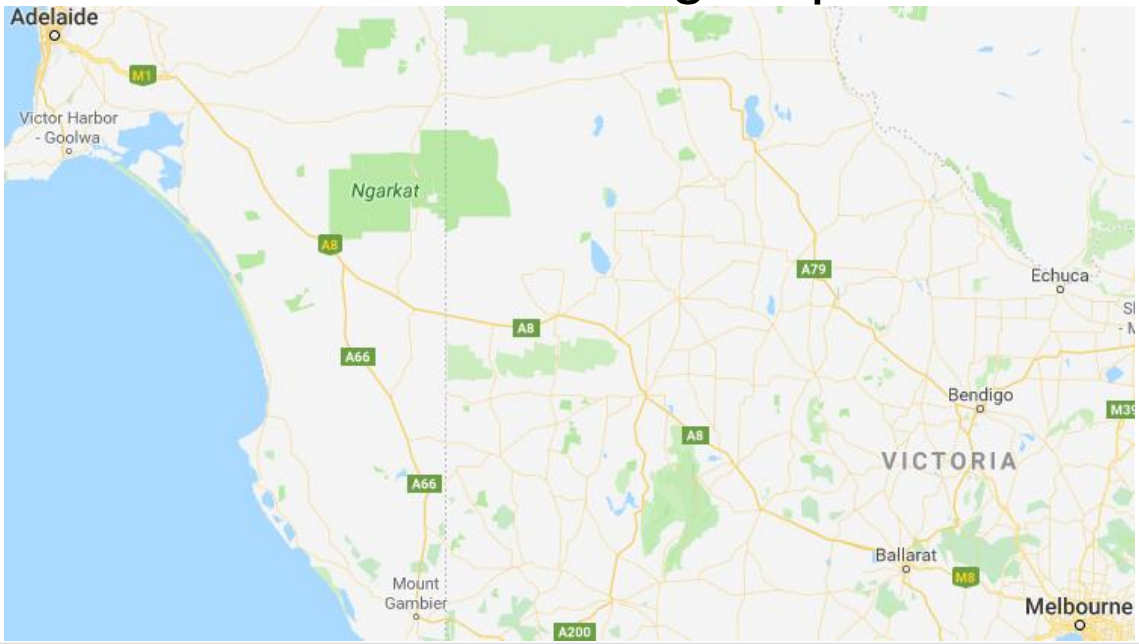
Programs and Algorithms

- What is a program?
  - A solution implemented with a specific programming language to solve a computational problem
- What is an algorithm?
  - The general solution designed for a specific problem
  - Specifies a sequence of step-by-step instructions with the flow of control indicating how each of the instructions should be executed

Let's do some thinking…

- How would you get to Adelaide from Melbourne by car?
- Which route would you take to get to the destination if there are multiple possible routes?
- What factors do you consider when choosing a specific route?

What is your algorithm?



A bit more thinking…

- How would you efficiently search a number in the list?
- Given a sorted list [1, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59], we want to search the position of number **37.**

Is there an algorithm for this problem?

Four Essential Components of a Program

- Input:
  - *Data* read in from an input device (e.g. keyboard, input file) needed to solve a computational problem
- Output:
  - *Result*s from the computation that presented on an output device (e.g. terminal screen, output file)
- Operations:
  - Arithmetic and logical operations specified as *program instructions* for solving a computational problem
- Control structures:
  - Program constructs that determine the *flow of execution*

Concepts of Abstraction

- Notion of abstraction:
  - Intends to ignore irrelevant details but focusing only on the essential ones
- When implementing a solution for a real-world problem:
  - You are only interested in modelling the essential properties of the solution rather than accommodating every single detail of the solution

We attempt to separate the logical aspect from the physical aspect of a computational problem and its solution as a realisation of abstraction.

An Example of Abstraction

- To implement a solution for a problem as a computational program, we need to know how to model:
  - The input (i.e. data needed for solving the problem)
  - The specific operation(s) needed to solve the problem
  - The output (i.e. result to be presented)

- To implement a solution for an additional problem as a Python program:

```
a = 1
b = 2
result = a + b
print("The addition of a and b is", result)
```

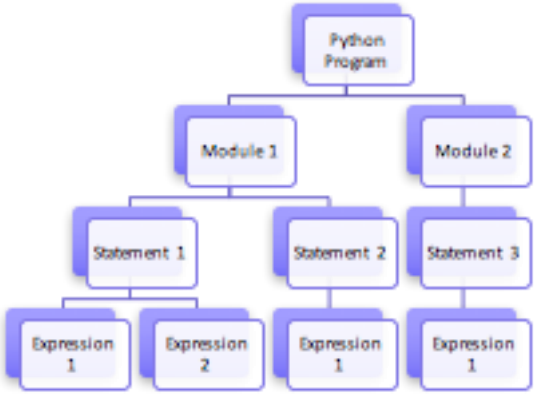
Basic Elements of Python: Objects and Variables

Syntax and Semantics

- Each programming language is defined by its own syntax and semantics
- Syntax:
  - A set of rules that defines how program instructions are constructed from various symbols and structures of a specific language
  - Programs constructed with invalid syntax will cause *syntax errors*
- Semantics:
  - Meaning associated with a program or its individual instructions that defines what the program is intended to achieve
  - Semantic errors (*logic errors*) happen when a program is syntactically well formed but did not produce the expected result

The Composition of a Python Program

- A Python program contains one or more modules (i.e. Python source files)
- Each module contains one or more statements
- Each statement contains one or more expressions
- Each expression is composed of Python objects and operators



Objects in Python

- Objects:
  - Core elements that Python programs manipulate on
  - Pieces of memory locations in the computer that containing (holding) a specific type of data value or literal

- Each object is associated with a specific data type (or object type)
  - How a program can manipulate it?
  - What kind of operations that a program can perform on it?

Examples of Python Objects

- How many Python objects in this program?

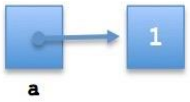
```
a = 1
b = 2
result = a + b
```

  - Both literal values ‘1’ and ‘2’ are objects of type *integer* (`int`)
  - The sequence of characters “The addition of a and b is” is another object of type *string* (`str`)

- To find out the type of an object with the built-in Python function `type()`:
  - E.g. `type(1)` or `type(result)`

Variables in Python

- Variables:
  - A means of associating a name to a value stored in the computer memory
  - Python: a name is a *reference* to a object
  - E.g.: `a = 1`



- A same variable can be associated with a number of different objects that could have a different value or a different data type

```
a = 1
a = 100
a = "Hi Python"
```

Python variables do not have a data type but the associated objects do; and the type of the object is determined by the literal that it contains.

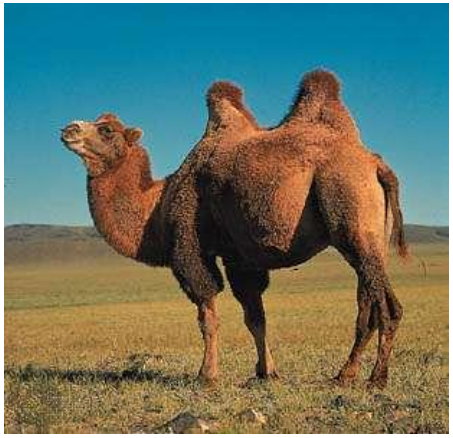
Naming Rules in Python

- Variable names in Python:
  - Can only contain: lowercase letters (a-z), uppercase letter (A-Z), digits (0-9), underscore (_)
  - Case sensitive
  - Cannot begin with a digit
  - Cannot be keywords (reserved words) in Python

and	as	assert	break	class	continue
def	del	elif	else	except	finally
for	from	global	if	import	in
is	lambda	nonlocal	not	or	pass
raise	return	try	while	with	yield
False	None	True			

Naming Conventions in Python

- When using multiple words as variable names:
  - Use a single underscore (_) as the *delimiter* between words (lowercase)
    - E.g.: `student_number, number_list`
  - Use the camelCase style
    - E.g.: `studentNumber, absentStudentNumber`



Use either one and be consistent throughout your programs

Variable names should be ***meaningful*** and usually *self-explained* with the kind of data that they represent (e.g. `a` vs. `studentNumber`?)

Naming Conventions in Python

- Python Enhancement Proposals (PEP)
  - "*Variable name should be lowercase, with words separated by underscores as necessary to improve readability.*"
  - https://www.python.org/dev/peps/pep-0008
- Google Python Style Guide
  - "*Function names, variable names, and filenames should be descriptive; eschew abbreviation. In particular, do not use abbreviations that are ambiguous or unfamiliar to readers outside your project, and do not abbreviate by deleting letters within a word.*"
  - https://google.github.io/styleguide/pyguide.html
- A Neural Model for Method Name Generation from Functional Description
  - 26th IEEE International Conference on Software Analysis, Evolution, and Reengineering, 2019

Week 1 Summary

- So far, we have discussed:
  - Python as a high-level programming language
  - Execution of Python programs
  - Concepts of program, algorithm and abstraction
  - Python object and variables

- Next week:
  - Core data types in Python
  - Operators and expressions
  - Statements and assignments
  - Standard input and output in Python

Reminder: Practical classes begin this week.