Lecture 1: Introduction

Dr Simon D'Alfonso

School of Computing and Information Systems
Faculty of Engineering and Information Technology

Today

1. Introductions. Subject information.
2. What is this subject about?
    1. Computers
    2. Algorithms
    3. Python programming language
3. Introduction to Python
4. Tools we will use
    1. IDLE
    2. Python files
    3. GROK learning platform

Contact hours

- The 2-hour lecture slot is on Tuesdays 2:15pm - 4:15pm.
- Each student is to sign up to one 1-hour weekly practical workshop/tutorial run by the subject tutors. Some tutorial options will be in-person/on campus and some will be online via Zoom.
- A weekly consultation hour with the subject lecturer will be held online each Friday at 3pm via Zoom. You can optionally attend when you require assistance with subject content.

Zoom Rules

- Mute your mic
- Use Chat to answer questions
- Ask a question related to the lecture by:
  - Posting in Chat
  - Raising your Hand in Zoom

Introductions: people

Dr Simon D'Alfonso
Lecturer
dalfonso@unimelb.edu.au

Mark Jiang
Head Tutor
yujing.jiang@unimelb.edu.au

Haoyu Yang
Tutor
haoyu.yang1@unimelb.edu.au

Dr Hayden O'Sullivan
Tutor
hayden.osullivan@unimelb.edu.au

LMS

See Canvas LMS (lms.unimelb.edu.au) for:

- Subject information
- Lecture slides, tutorial sheets and example programs
- Recordings of content explanations, tutorial solutions and Grok program solutions
- Announcements
- Discussion forums

Assessment

- Mid-semester test (10%), run as LMS quiz around week 7
- Assignments (30%):
  - Assignment 1 (10%) , due around week 8
  - Assignment 2 (20%) , due around week 12
- Final exam, (60%) a 3-hour summative assessment of all coursework, TBA via LMS announcement
- Hurdle requirement: 30/60 for final exam

We want to support you

We are here to help. If you need support, please reach out.

- Ask your tutor
- Form or join a study group
- Attend consultations (Friday 3pm via Zoom)
- Post your question in LMS discussion forum
- Email us

Why learn programming?

Computer programming is empowering. For starters, being able to tell a machine with significant processing power to calculate or solve some problem for you is handy to say the least. Beyond giving machines instructions and getting results, programming also teaches you abstract thinking and problem-solving. In fact, troubleshooting, problem-solving skills and creativity are key to becoming a successful computer programmer.

What you will be studying

- Data structures
- Computational problem solving
- Designing, implementing, testing and debugging programs that use fundamental programming constructs such as:
  - Basic input/output
  - Variables
  - Mathematical, logical and string operators
  - Conditionals and iterative structures
- Functions
- This subject is designed for students who have little to no background in programming

What is a computer program?

- A set of instructions given to a computer for it to execute a task
- Programming involves
  - Understanding the task to be carried out
  - Designing a precise process to carry out the task
  - Coding: writing instructions to the computer in a programming language for how to execute this process
  - Testing the code to see that it performs the task correctly

Code example

```
principal = int(input('Enter the initial principal: '))
rate = float(input('Enter the annual interest rate: '))
years = int(input('Enter the number of years: '))
rate = rate/100
amount = principal

for year in range(years):
    amount = amount * (1 + rate)

print('The final value is:', round(amount))
```

- Note how code is made up of STATEMENTS which are executed by the computer in order.
- What does this code do?

Commenting the code

```
# a program to calculate the future value of a sum based on the
initial principal, the annual percentage interest rate & number of
years


#get data from the user
principal = int(input('Enter the initial principal: '))
rate = float(input('Enter the annual interest rate: '))
years = int(input('Enter the number of years: '))
#convert given rate to a fraction, and set calculated amount as
principal amount to start with
rate = rate/100
amount = principal
#for each year, increase the currently calculated amount by the rate
for year in range(years):
    amount = amount * (1 + rate)
#report back the final value
print('The final value is:', round(amount))
```

Approach Comparison

```
principal = int(input('Enter the initial principal: '))
rate = float(input('Enter the annual interest rate: '))
years = int(input('Enter the number of years: '))
rate = rate/100
amount = principal

#directly using the compound interest formula instead
final_value = amount * ((1 + rate) ** years)
print("The final amount is:", round(final_value))
```

What makes a piece of code good?

- effective - does the task correctly
- efficient - does the task with the minimum amount of code, or computer time, or user time and effort
- obeys the syntax of the language
- easy to understand (by humans)
- easy to maintain
- easy to re-use

Syntax

- The syntax of a computer language is the set of rules that defines the combinations of symbols that count as correct statements/expressions in that language
- There are many programming languages, each with their own syntax. Python, JavaScript, C, C++, PHP are but a few examples
- A statement that works in one language need not work in another language (e.g., Python vs. PHP)
- The syntax available for a language might make it suited to certain tasks. Python in particular has many neat and useful syntactic constructs, such as list comprehensions

Natural Language vs Programming Language

- Natural languages
  - complex but flexible syntax
  - meaning relies on context
  - ambiguity is resolved by the listener
  - missing or wrong parts might not matter too much
  - extensive vocabulary, with subtle differences in meaning
  - is understood at the speed of conversation
- Programming languages
  - strict and precise syntax
  - simpler, well-formed and structured vocabulary
  - meaning must be clear and precise
  - missing or wrong elements lead to a complete breakdown
  - understood and acted on VERY QUICKLY in milliseconds or less

Natural Language vs Programming Language

- Computer programs are limited in terms of things that we humans process easily as natural language agents.
- Take the following inference:
  - Nothing is better than winning the lottery
  - A black and white television is better than nothing
  - Therefore, a black and white television is better than winning the lottery
- Directly translating this into computer code would end up with the computer saying this inference is valid:
  - better_than('nothing', 'lottery').
  - better_than('black_white_tv', 'nothing').
  - better_than(X, Z) :- better_than(X, Y), better_than(Y, Z).

Natural Language vs Programming Language

- Computers don't tolerate imperfection.
- Take the following inference
  - Morning star = Venus
  - Evening star = Venus
  - Therefore, Mornin star = Evening star
- In computer code, this would be written as something like

```
>>> morning_star = 'venus'
>>> evening_star = 'venus'
>>> morning_star == evening_star
True
```

- What about

```
>>> morning_star = 'venus'
>>> evening_star = 'venus'
>>> mornin_star == evening_star
```

Comments and pseudo code

- Comments (which we saw earlier) help bridge code to natural language
- Pseudocode is a plain language description of the steps in an algorithm or computer program. Pseudocode often uses structural conventions of a normal programming language (such as IF-THEN-ELSE or FOR LOOPS) but is intended for human reading rather than machine reading.
  - https://towardsdatascience.com/pseudocode-101-an-introduction-to-writing-good-pseudocode-1331cb855be7
  - https://www.povertyactionlab.org/sites/default/files/research-resources/rr_datacleaning_Pseudocode.pdf

Algorithms

An algorithm is a sequence of steps for a task, or part of a task, that will work for every situation that could be encountered:

- Solves a general class of problems
- Consists of a finite number of instructions
- Each individual instruction is well defined
- Describes a process that eventually halts after arriving at a solution to a problem

Algorithm Example 1

- Suppose we have two separate collections of beans. How can we determine which collection of beans has the greater quantity?
- As humans with visual capability, we might see that it is sufficiently obvious that one collection is bigger than the other (e.g., 5 beans versus 50 beans)
- Or as humans with mathematical counting ability, we might simply count the numbers of beans in each to determine which has the greater quantity (or equal).
- However, what about an algorithm for this determination, that does not involve seeing or counting?

Algorithm Example 1

- A simple algorithm to achieve this determination is as follows.
  1. Remove one bean from collection 1
  2. Remove one bean from collection 2
  3. If collection 1 has no beans left and collection 2 has no beans left, they are of equal quantity
  4. If collection 1 has no beans left and collection 2 has beans left, collection 2 is bigger
  5. If collection 2 has no beans left and collection 1 has beans left, collection 1 is bigger
  6. If both collections have beans left, repeat steps 1 - 6

Algorithm Example 2

- It will generally be the case that there is more than one algorithmic way to approach solving a task.
- Sorting collections of things is a commonplace task.
  - A comparison measure is required: less than, greater than, equal to.
  - One example is sorting a list of numbers from highest to lowest. Uses simple numerical comparison (e.g., 1 < 2, 4 > 3, 5 = 5).
  - Another example is sorting words in alphabetical order. Uses character comparison (e.g., a < b, z > c, a = a).

Algorithm Example 2

- Sort the following from lowest to highest
  - 5,4,8,7,1,2,2,9
  - zebra, lion, tiger, antelope, aardvark
- How would an algorithm do this in a well-defined, instruction-based way?
- There are many types of sorting algorithms: selection sort, insertion sort, heap sort, merge sort, etc.
  - Selection sort: https://www.youtube.com/watch?v=g-PGLbMth_g
  - Insertion sort: https://www.youtube.com/watch?v=JU767SDMDvA

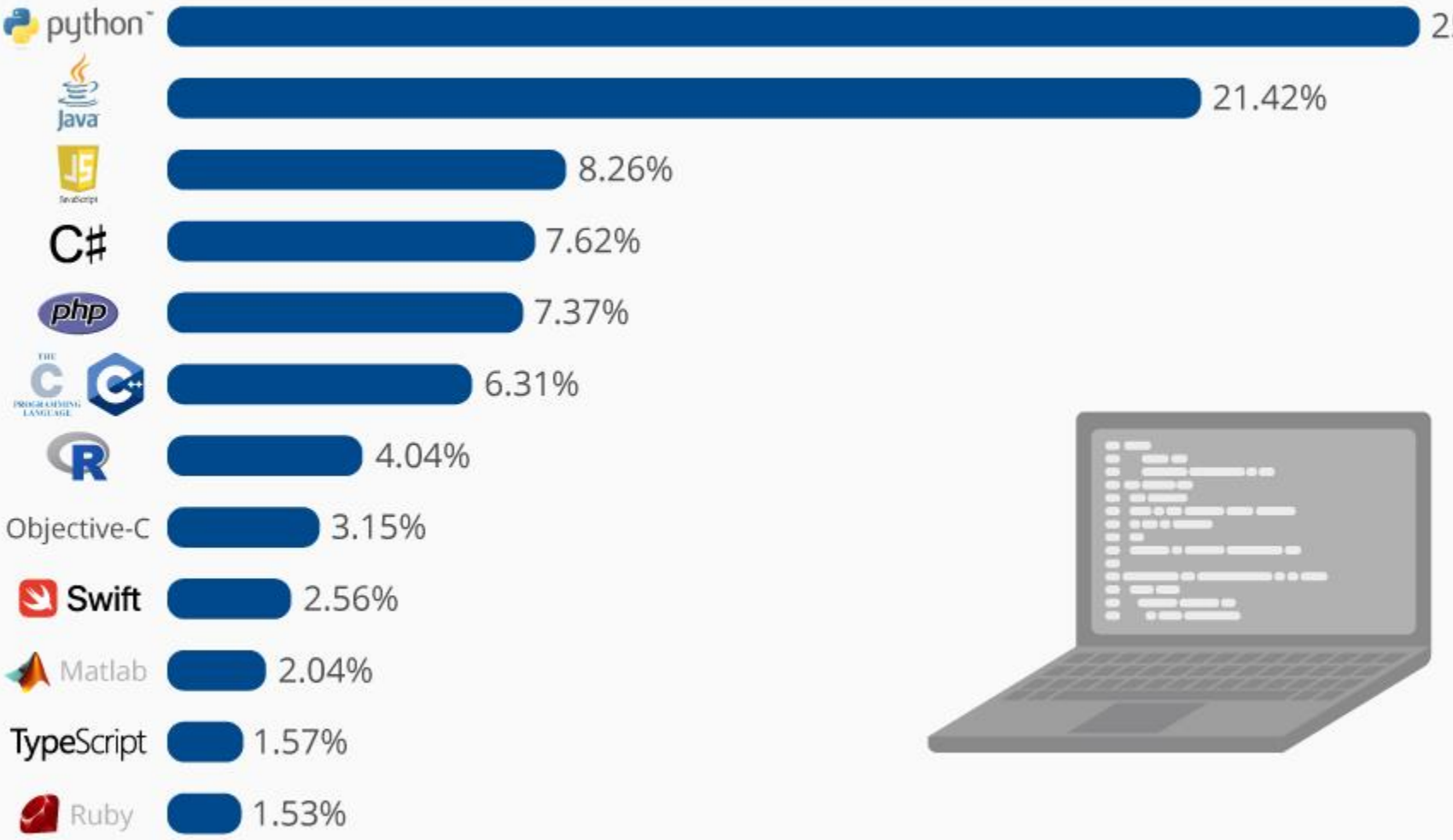
Python

Python

- Created by Guido van Rossum in the late 1980s and early 1990s, Python ranks as one of the most popular programming languages.
- It is an interpreted high-level general-purpose programming language.
- Interpreted versus compiled
  - Compiled languages are converted directly into machine code that the processor can execute. As a result, they tend to be faster and more efficient to execute. They also give the developer more control over hardware aspects, like memory management and CPU usage.
  - Interpreted languages are not compiled into the instructions of the target machine (machine code) but are instead interpreted by an interpreter (written at lower level) that then does the job of converting the code to instructions of the target machine.
- High-level means that Python is distanced from the low-level, implementation details of the computer.
  - Rather than having to write technical code that directly controls computer elements such as memory, high-level languages abstract from these details.
  - High-level programming languages such as Python are clear, generally easier to write in and closer to natural language.
- Procedural/imperative
- Python can be used for many purposes

The Most Popular Programming Languages

Share of the most popular programming languages in the world*



* Based on the PYPL-Index, an analysis of Google search trends for programming language tutorials.

Source: PYPL

More about Python

- Python is free
- It has a supportive community
- It has an established ecosystem
- Python programming has low barriers to entry
- Extensive libraries and frameworks
- Very versatile
  - AI and machine learning
  - Data analytics
  - Data visualization
  - Scientific programming
  - Web and desktop applications
- Some useful sites
  - https://www.w3schools.com/python/
  - https://www.tutorialspoint.com/python/index.htm
  - https://realpython.com/

Install Python

- We will use Python v 3.6 or later
- Get a copy of Python for your computer. There are free versions for Windows, MacOS and Linux at https://www.python.org
- IDLE is Python's Integrated Development and Learning Environment. It allows programmers to easily write Python code and can be used to execute a single statement and create, modify, and execute Python scripts.
- Programs can also be written in text files and executed via the operating system command line.

Grok Learning Environment

- GROK is an online platform that provides an environment to learn Python with structured modules.
- All you need to access the system is a browser, an Internet connection and your GROK/unimelb account
- The GROK link for this subject is https://groklearning.com/course/unimelb-comp90059-2022-s1/

Some first steps with Python

Print()

The ‘print’ function is how we tell Python to show something on the screen.

```
>>> print('hello')
hello
>>> print('Python')
Python
>>> print(3)
3
>>> print('3')
3
```

Input()

The ‘input’ function puts a prompt message on the screen, and then gets the value of the string that the user enters via the keyboard, which they finish by pressing the enter/return key.

```
name = input('Enter your name: ')
print(name)
```

```
Enter your name: Simon
Simon
```

Variables, Variable Types

- A variable is something we create in a program to store a value under the name we give it.
- string variable – a sequence of characters

```
name = 'Katie'
print(name)
>>> Katie
```

- integer variable – a whole number

```
price = 17
print(price)
>>> 17
```

- Variables are of different types depending on the kind of data they store.

Variable Behaviour

variable types behave differently

Integers

- price = 17
- print(price)
- **print(price + 3)**
- **print(price * 3)**
- print(price / 3)
- print(price // 3)
- print(price % 3)
- print(price ** 2)
- new_price = price + 3
- print(new_price * 2)

Strings

- name = 'Katie'
- print(name)
- **print(name + name)**
- **print(name * 5)**
- new_name= name + ' '
- print(new_name* 3)
- **print(name - name)**
- **print(name + 1)**

```
#Say Your Name A lot
name = input('Enter your name: ')
print(name)
print(name, name)
print(name * 5)
name = name+"T"
print(name * 5)
```

```
#Do some simple sums
string1= input('Enter a number: ')
string2= input('Enter another number: ')

number1= int(string1)
number2= int(string2)

sum = number1+ number2
product = number1* number2
ratio = number1/ number2
power = number1** number2

print('Sum = ', sum)
print('Product = ', product)
print('Ratio = ', ratio)
print('Power = ', power)
```

Academic honesty

- This subject is run in accordance with the University's Academic Honest and Plagiarism Policy (https://academicintegrity.unimelb.edu.au):

Academic work submitted for assessment or publication must be the original work of the author or authors. If the ideas or words of others have been drawn upon, this must be thoroughly and clearly acknowledged using agreed scholarly conventions.

- All examinable work (including the two GROK programming assignments) that you submit for this subject must be your own.
- Common attempts to escape undetected are:
  - changing the comments but not the code
  - changing variable names
  - rearranging blocks of code (sometimes breaking the logic in the process!)
  - It is all too easy to automatically pick up on all of these, and many, many more, approaches using software plagiarism detection software ... and we do check

Academic Honesty

So, what is appropriate?

- You are encouraged to share/collaborate directly on code for any non-examinable items (notably the worksheet questions) ... and you will learn a lot from reading the code of others (including the sample solutions in the worksheets)
- You are very welcome to discuss with fellow classmates your approach to worksheet questions and projects, in non-specific, conceptual terms

Wally's Four Tips

- Learn each step well before you proceed
  - Keep up with lectures and workshops
  - If confused, go back to re-learn earlier concepts
- Enjoy being confused
  - Struggle to understand. Keep puzzling things out, looking things up, and asking questions
- Be sure about your basic math and logic
- Play
  - Experiment with Python in IDLE, Grok and text files
  - Think up your own program ideas and try to build them
  - Find Python tasks/challenges on the Web

Things to do before next class

- Install Python on your computer. Enter and run the programs from this lecture into IDLE or via command line program file
- Log on to GROK and try the module ‘Worksheet 0: Building blocks’
- Check out the resources about Python listed in CANVAS
- Workshops start in Week 2

Events Confirmed	Week	Date, Time, Location
Summerfest Welcome Back Expo	1	1-Mar-22, 2-5pm 2-Mar-22, 2-5pm 3-Mar-22, 2-5pm 4-Mar-22, 2-5pm
Welcome event with DSSS & HackMelbourne	2	7-Mar-22, 11:30am
Google Career Discussion	2	8-Mar-22, 5 - 6:30pm
Spaghetti Bridge Building Comp Collab	2	11-Mar-22, 3-5:30pm
Robogals collab industry week	3	this whole week
Google Women in Tech clubs session	3	16-Mar-22, 5-5:45pm
Macquarie Grad Program Event	3	TBD
...	...	...

Events planned (wk4 onwards):

- Annual Hackathon
- Workshops for CV and job seeking
- High Tea networking
- Industry Panels
- Free lunch on campus
- Study sessions
- Tech-related clubs collaboration
- SWOT cake
- ...

Other opportunities:

- Weekly job posts
- Tech-related events/competitions
- Volunteering opportunities
- ...

WOMEN IN TECH

The University of Melbourne

JOIN US HERE!

Lecture Identification and Acknowledgement

Coordinator / Lecturer: Simon D'Alfonso

Semester: Semester 1, 2022

These slides include materials from 2020 - 2021 instances of COMP90059 run by Kylie McColl or Wally Smith