Basics of Computer Programming
& Software Engineering.
ENGF0002: Design and Professional Skills

Prof. Mark Handley
University College London, UK
Based on slides from George Danezis

Term 1, 2019

Introduction

Introducing Basics.

In the 'Basics' topic we will:

- Introduce basic Python **language** concepts.
- Illustrate **algorithms**, to perform numerical computations.
- Expose you to the problems of **correctness**, and **testing**.
- Introduce **good practices** when programming.

What is Computer Science & Software Engineering?

Computer Science:

- Studies the nature of **information, computation, complexity** of algorithms, and their **correctness**.
- Deep mathematical foundations, including logic, algebra and probability theory.

Software Engineering:

- Studies how teams can repeatably **build high quality, correct, usable and efficient software**, to meet **people's needs**.
- Programming is foundational and an important part of it.

The two are interlinked, and we will study them together.

Programming & languages (I).

Programming & Programmability:

- A device is programmable if it allows a programmer (developer, engineer) to **alter its behavior**. This is the act of programming.
- We usually think of a device as programmable, when the **program itself takes the form or information**, rather than physical modification.
- A computer is the **ultimate programmable device**, and can execute all computations. Simpler ones include video recording devices, microwave ovens, and alarm clocks.
- **Programming is specializing** a device to solve **a problem that people have**.

Programming & languages (II).

Programming Languages:

- The **information describing the program** (code), is expressed in a programming language.

- **Trivial** programming languages: button pressed to program an alarm clock. They are very **low-level ways**, and lead to errors and inflexibility.

- Full computers are programmed in **higher-level formal languages**. Programmers may express their intent directly, build complex abstractions, and compose programs from smaller fragments.

- Programming languages are formal languages but also **human languages**. Programmers, use them to **express and communicate their intent**.

Why Python?

- **Real-world**, widely used programming language
  (3rd in 2019 TIOBE ranking.)
- **Multi-paradigm**: scripting, procedural, functional elements, object oriented, generics.
- Extensive **eco-system** of tools and libraries. Great documentation.
- Significant **industrial uses**.
- **High-productivity**, and perfect for rapid prototyping.

A **weakness** we will turn to a strenth:

- It does not force you to adopt **good programming practices**.
- It **supports them**, but you must learn them!

Other languages you might want to learn.

Different problems may require **different tools**:

- Lower-level: Rust∗, C++, **C** — COMP0002.
- Static typing: Go∗, **Java**, C# — COMP0004.
- Web: Javascript∗.
- Functional: **Haskell**, Scala∗ — COMP0002.

∗ - trendy languages at the moment

All programming languages mix and match a few principles.
Good computer scientists **know those principles**,
and can **work in any language**.

Using Python interactively.

- Python 3.7.
  (https://www.python.org/about/gettingstarted/)
- Open a command line console, and run the Python interpreter.
- Type your fist command:
  ```python
  print("Hello World!")
  ```
- You should see it executing!
  ```
  mjh$ python
  Python 3.7.2 (default, Dec 30 2018, 08:59:00)
  [Clang 9.1.0 (clang-902.0.39.2)] on darwin
  Type "help", "copyright", "credits" or "license" for more information
  >>> print("Hello World!")
  Hello World!
  >>>
  ```

Key resources and tools.

The interactive interpreter is only good for quick experimentation.

- **Code editor**: atom (hip!), Sublime (hip!), Visual Studio Code.
  Must haves: Good syntax highlighting, good handling of files & folders, whitespace, looks very cool.
- **Browser** with Python **Documentation**:
  `https://docs.python.org/3/`.
  (including Python **tutorial** and **Library reference**.)
- **Command line** & learn how to use it.
  `https://www.lynda.com/Linux-tutorials/`
  `Learn-Linux-Command-Line-Basics/435539-2.html`
- **Stack Overflow** for Q&A.
  `https://stackoverflow.com/`
- **Install pytest**. `https://docs.pytest.org/`

Your physical & mental well being.

- Approach the physical activity of programming with **professionalism**.
- Think of the **ergonomics** of your physical environment: chair height, desk, posture, keyboard style, monitor type and positioning.
- Make sure you **enjoy your working environment**: light, sounds, distractions.
- **Take breaks**, at least every hour.

The 'Hello World!' program in Python.

Programs live in files. A simple Python program:

```python
# File: hello_world.py

def hello_world():
    print("Hello World!")

# run if called as `python hello_world.py'
if __name__ == "__main__": # Python idiom
    hello_world()          # `Call' the function `hello_world'
```

Execute your first program, by running:

```
$ python3 hello_world.py
Hello World!
```

Comments

Comments are not executed by Python, but form part of your program.

- Comments **communicate intent** to your future self and others.
- Express in comments concepts you cannot directly express in code.
- Prefer to express concepts directly in code.
- Keep comments local to the relevant code.
- Do not overcomment; avoid duplicate, redundant or wrong comments.
- Every comment will have to be maintained in the future.

Whitespace.

Python is **sensitive to indentation** using whitespace. Use 3 or 4 spaces (not a tab) to indent blocks of code. More about this later . . . .

```python
# File: hello_world.py

def hello_world():
    print("Hello World!")

# run if called as `python hello_world.py'
if __name__ == "__main__": # Python idiom
    hello_world()              # `Call' the function `hello_world'
```

How do we know a program does what it should?

Correctness is the most important problem in software engineering and computer science!

Testing is the key technique to produce high quality, correct programs. It is an activity that is continuous, and performed in parallel with programming.

Complimentary techniques for ensuring program correctness include **formal verification**. Those are more expensive, but necessary to reason about very complex problems.

Finally, formal periodic **code reviews**, or continuous reflection through **pair programming** also improve quality.

Test Code Continuously.

Testing is the most important technique to gain confidence a program does what it should:

- Unit Test every 2-5 lines of code you write. Think how to **test every snippet of code** just before, while, or just after you write it.
- Use a mature **tool for testing** your programs. We will use `pytest`. `https://docs.pytest.org/`.
- **Different testing techniques** for code correctness, integration, security, performance and user experience.
- Shortcomings of testing: **lack of completeness**.

Testing the 'Hello World!' program.

Tests are snippets of code executing parts of your program.
Eg. the test of the simple Python program is:

```python
# File: test_hello_world.py
from hello_world import hello_world

def test_hello_world(capsys):
    hello_world()
    out, _ = capsys.readouterr()
    assert out == "Hello World!\n"
```

We run the **test suite**, by executing `pytest test_hello_world.py` on the command line.

Code documentation and its tests.

Code documentation is not executed, but forms part of the program. It is useful to your future self, or others that want to use part of your program. Hence, **document reusable units of code**.

```python
# File: hello_world_doc.py

def hello_world():
    """ Print a welcoming message to stdout.

        >>> hello_world()
        Hello World!
    """
    print("Hello World!")
```

You must **test code in documentation**.
`pytest -vs --doctest-modules test_hello_world.py.`

Example of a successful test run.

Executing `pytest` runs all tests in functions named `test_*` and documentation strings.

```
$ pytest -vs --doctest-modules src/*hello*.py
==================================== test session starts ====================
platform darwin -- Python 3.7.0, pytest-3.7.4, py-1.6.0, pluggy-0.7.1 -- /opt/loca
cachedir: .pytest_cache
rootdir: /Users/mjh/teaching/engf0002/Design_and_Professional_Skills/Topics/01_Bas
collected 2 items

src/hello_world_doc.py::hello_world_doc.hello_world PASSED
src/test_hello_world.py::test_hello_world PASSED

================================= 2 passed in 0.02 seconds ===================
```

Summary and next steps.

The rhythm-of-the-programming-business:

- **Think** of the problem → **Code** feature (2-5 lines) → **Write test** for feature → Run **all** tests → Fix until **all tests pass** → Think . . .
- Working programmers **run tests a few times per minute**.
- Testing practice imposes an **incremental approach** to software building.
- Professional standards: aspire to deliver programs with **zero bugs**. Remember that **'bugs' may cost millions or even kill**.

Still to cover: functions and function calls (`def`, `()`), modules (`from`, `import`) and a lot more python . . .

Algorithms

What is an 'algorithm'?

An algorithm describes a sequence of steps leading to the solution of a computational problem.

You are familiar with a number of algorithms from maths:

- How to perform long multiplication and division with a pencil and paper.
- How to find the roots of a quadratic equation $ax^2 + bx + c = 0$.
- How to expand brackets for $(a + b)(c + d) =$?

Beyond maths . . .

Once we express wider parts of the world as information, algorithms can solve more exciting tasks: eg. rendering a 3D scene in a computer game, encrypt communications, build interactive social networks on-line, and program cars to drive autonomously.

Lifts

Problem Statement

A building has a lift.
Inside the lift are buttons to choose floors.
Outside the lift are buttons to call the lift.
How should the lift choose which floor to go to next?
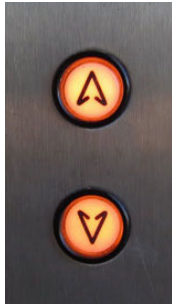
Lifts

Algorithm 1

Go to the nearest floor that is selected.
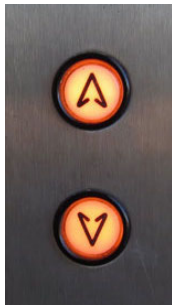
Algorithm 1

Go to the nearest floor that is selected.

Bug report

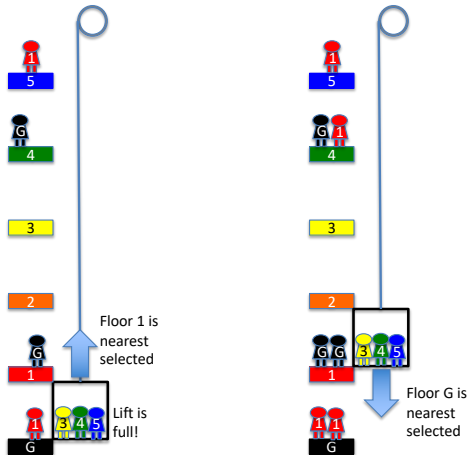- Under certain workloads, people in the lift starve to death before they reach the destination floor.
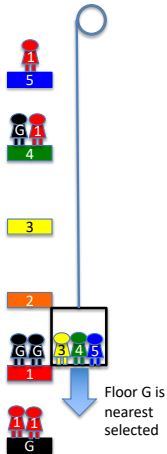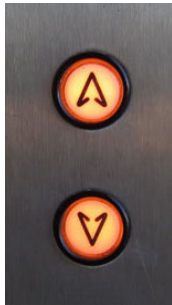
Algorithm 1

Go to the nearest floor that is selected.

Algorithm 1

Go to the nearest floor that is selected.

Floor 1 is nearest selected

Lift is full!

Floor G is nearest selected

Algorithm 1

Go to the nearest floor that is selected.

Floor 1 is nearest selected

Lift is full!

Floor G is nearest selected

People are getting pretty grumpy up here

Floor 1 is nearest selected

"We never get above floor 1!"

Algorithm 1

Go to the nearest floor that is selected.

Bug report

- Under certain workloads, people in the lift starve to death before they reach the destination floor.

Diagnosis

Need to prioritse going to the floor selected by people in the lift.
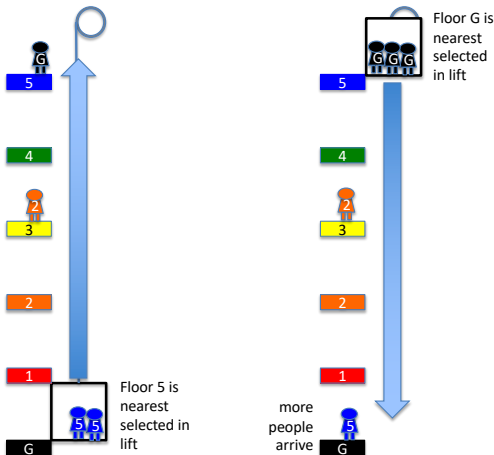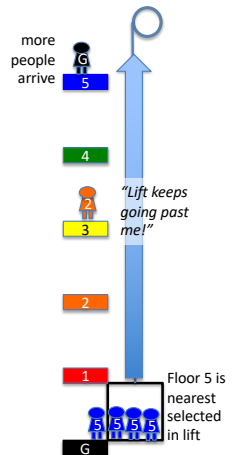
Lifts

Algorithm 2

- Go to the nearest floor selected by people in the lift.
- If no floor selected in the lift, go to nearest floor where someone has pressed a call button.
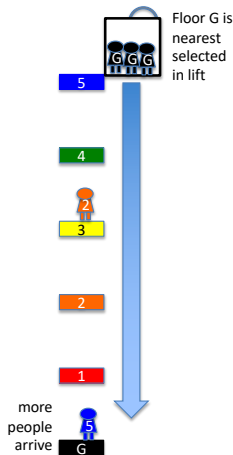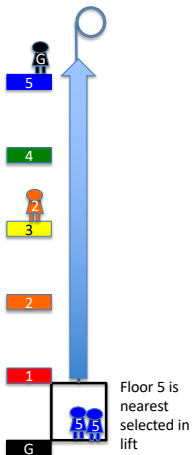
Algorithm 2

- Go to the nearest floor selected by people in the lift.
- If no floor selected in the lift, go to nearest floor where someone has pressed a call button.

Bug report

- Under certain workloads, people waiting for the lift starve to death.

Algorithm 2

- Go to the nearest floor selected by people in the lift.
- If no floor selected in the lift, go to nearest floor where someone has pressed a call button.

Floor 5 is nearest selected in lift

Algorithm 2

- Go to the nearest floor selected by people in the lift.
- If no floor selected in the lift, go to nearest floor where someone has pressed a call button.

Floor 5 is nearest selected in lift

Floor G is nearest selected in lift

more people arrive

Algorithm 2

- Go to the nearest floor selected by people in the lift.
- If no floor selected in the lift, go to nearest floor where someone has pressed a call button.

Algorithm 2

- Go to the nearest floor selected by people in the lift.
- If no floor selected in the lift, go to nearest floor where someone has pressed a call button.
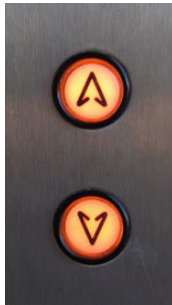
Revised Bug report

- Under certain workloads, people waiting for the lift starve to death because the lift always goes past their floor.

Diagnosis

Should not skip a floor where someone wants to go in the direction the lift is going.

Algorithm 3

- Go to the next floor selected by people in the lift, but don't go past a floor where someone wants to go in the same direction the lift is going.
- If no floor is selected, go to the nearest floor where someone has pressed a button.
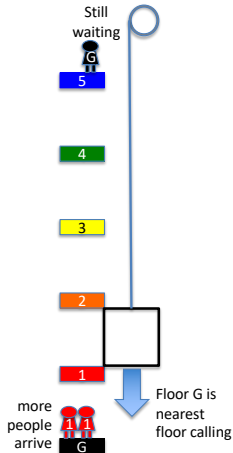
Algorithm 3

- Go to the next floor selected by people in the lift, but don't go past a floor where someone wants to go in the same direction the lift is going.
- If no floor is selected, go to the nearest floor where someone has pressed a button.

Bug report

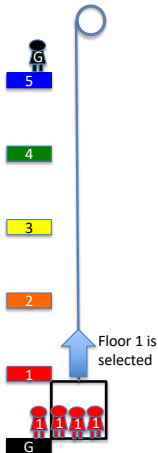- Under certain workloads, people waiting for the lift starve to death.

Algorithm 3

- Go to the next floor selected by people in the lift, but don't go past a floor where someone wants to go in the same direction the lift is going.
- If no floor is selected, go to the nearest floor where someone has pressed a button.
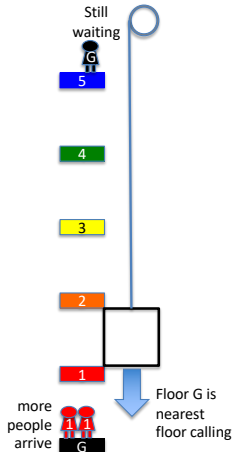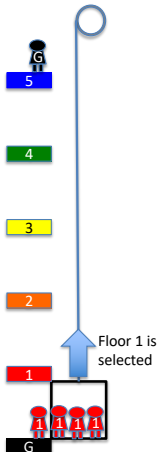

Floor 1 is selected

Algorithm 3

- Go to the next floor selected by people in the lift, but don't go past a floor where someone wants to go in the same direction the lift is going.
- If no floor is selected, go to the nearest floor where someone has pressed a button.

Algorithm 3

- Go to the next floor selected by people in the lift, but don't go past a floor where someone wants to go in the same direction the lift is going.
- If no floor is selected, go to the nearest floor where someone has pressed a button.

Still waiting

Floor 1 is selected

Floor G is nearest floor calling

more people arrive

Tired of waiting

Floor 1 is nearest selected

Algorithm 3

- Go to the next floor selected by people in the lift, but don't go past a floor where someone wants to go in the same direction the lift is going.
- If no floor is selected, go to the nearest floor where someone has pressed a button.

Revised Bug report

- Under certain workloads, people waiting for the lift starve to death before it reaches their floor.

Diagnosis

Lift should not turn round if it's heading towards people who are waiting.

Algorithm 4

- Go in one direction, stopping at floors where:
    - people outside want to go in that direction, or
    - where people in the lift want to go in that direction.
- Change direction only when there are no more selected floors or call buttons pressed in that direction.
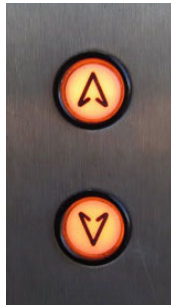
Algorithm 4

- Go in one direction, stopping at floors where:
    - people outside want to go in that direction, or
    - where people in the lift want to go in that direction.
- Change direction only when there are no more selected floors or call buttons pressed in that direction.

Bug reports

- — NO BUG REPORTS FILED —

Challenge 1

- What should the lift do if no call button is pressed?

Challenge 2

- Extend the algorithm so it works efficiently to control two lifts.