INFO1110/COMP9001 Introduction to Programming

Contents

Notes from the creator

About these notes

The following notes and examples are written to target Python version 3.4 and above.

Code blocks beginning with `%%bash` contain bash commands which can be run in the terminal.

Tips for doing well in INFO1110/COMP9001:

Although these notes supplement studies and understanding of concepts, learning to program requires constant practise. This is especially relevant to those new to programming. There is no substitute to taking the time to go through problems and coding yourself. I cannot stress this point enough.

1. **Don't be afraid to make mistakes whilst you're coding**

   If this is your first time coding, mistakes are inevitable. Don't be disheartened if your code doesn't work straight away or if a concept is hard to grasp. The key is to perservere, keep going and keep practising. Reframe failures as ways in

which you can learn and grow. Chances are that when you make a mistake, it's more likely you'll remember it and then never make the same mistake again.

2. **Ask for help when you need it**

   The staff are there to help you. If you don't quite understand a concept, don't just ignore it. Ask for help! It's better to be constantly asking for help rather than letting it pile up and realising before the final that you don't understand anything at all. If you're afraid of asking in person, you can use Ed, or research online. There are plenty of resources out there: Stackoverflow, YouTube, online tutorials, etc.

3. **Nail the fundamentals**

   All the concepts build upon each other. Make sure you have a solid grasp of the fundamentals. Remember that this is an introduction to programming, so you want to be building a good understanding of the concepts. A good base means that you are able to learn harder concepts and different languages faster.

4. **Keep up to date (this includes lecture content and tutorial work)**

   The workload builds up, so it's important to keep up to date. It's understandable that uni gets busy, but try at least to be up to date with the previous week's work before the next week starts. This way you aren't cramming right before a quiz or an exam.

5. **Go to the tutorials**

   The tutorials are a great place for learning and getting help if you need it. Important concepts are often reviewed and emphasised. There's also a weekly task that counts towards your attendance/overall grade. They're easy marks that you shouldn't waste - especially if you don't feel confident about the final.

6. **Do the best that you can throughout the semester and before the finals**

   A 60% weighted final is a lot of pressure on just one paper, so make sure your cumulative mark up until that point is the best that it can be. It places a lot less pressure on you and gives you that extra bit of confidence. Plus, doing well during the semester is a good indicator that you know your content, so that's always good.

7. **Practise! Practise! Practise!**

   If this final point is the only thing you take away from all these tips, I'd be happy. Remember that coding is something that requires time, work and practise. If you're a beginner programmer and you want to do well, you have to put in the effort. If you're struggling to understand a particular concept, do more problems on that topic. You aren't just limited to tutorial problems, there are so many sites online. If you want somewhere to get started, I recommend checking out

   - CodingBat, `https://codingbat.com/python`
   - HackerRank, `https://www.hackerrank.com/domains/python`
   - ProjectEuler, `https://projecteuler.net/archives`

Week 01

Course outline

Assessment timeline and weightings can be found on CUSP.

https://cusp.sydney.edu.au/students/view-unit-page/alpha/INFO1110

https://cusp.sydney.edu.au/students/view-unit-page/alpha/COMP9001

The recommended textbook is Sedgewick's *Introduction to Programming in Python: An Interdisciplinary Approach*. It is available for free through the University library website, or via academic sign-in at:

https://www.oreilly.com/library/view/introduction-to-programming/9780134076539/?ar

Introduction to computers

A program is a set of instructions that a computer can understand, and then execute. A compiler will translate a source code into machine code.

- Write source code – code which a human can read
- Compile the code to executable code – code which a computer can read
- The computer then runs the executable code

You can tell Python to compile and run your program with the `python3` command in the terminal.

To check which version on Python you are running, enter the following command in the terminal.

```
python3 --version
```

Using the terminal

Instead of navigating the folders (directories) on your computer using File Explorer on Windows or Finder on MacOS, you can navigate your computer using commands in the terminal. Here are the important ones to know.

Action	Command
To show your current folder	`pwd`
To list files in the current folder	`ls`
To make a new folder	`mkdir [DIRECTORY_NAME]`
To move into a folder	`cd [DIRECTORY_NAME]`
To go back to your home folder	`cd ~`

It may be helpful to remember these as follows.

- `pwd`: Print working directory
- `ls`: List files and directories
- `mkdir`: Make directory

- `cd`: Change directory

Text editors

There are many to choose from. Choose your favourite.

- Visual Studio Code (https://code.visualstudio.com/)
- Atom (https://atom.io/)
- Sublime Text (https://www.sublimetext.com/3)
- Nano
- Vim

The `print` function

The `print` function can be used to print things to the terminal. You need to invoke the print function with parentheses `()` and pass inside the parenthesis what you want it to print.

```
print("Hello, world!")
print(42)
```

```
Hello, world!
42
```

Variables and Expressions

Everything on your computer is stored as zeros and ones (binary), each zero or one is known as a bit. 8 bits make 1 byte.

A **variable** is memory space, reserved by a name (identifier). There are two types of variables:

1. Primitive: these use a fixed amount of memory
2. Object: memory used for objects can changed over time

   Note: In python, everything is an object. Objects are covered in more detail in week 6.

When working with variables:

1. DECLARE a variable
2. INITIALISE the variable
3. USE the variable

By declaring variables, You can now refer to the reserved memory space by its name. Give your variable a value by initializing it. If not initialized, the memory space will be filled by a default value (which differs between machines).

> Variables in python are function scoped. A variable declared within a function can only be used inside that function.

Expressions are a combination of variables and other items which can be evaluated, and will produce a value. We construct expressions using operators

Operators

The basic mathematical operators are

- + add
- - subtract
- * multiply
- / divide

```
print(6 + 4)
print(6 - 4)
print(6 * 4)
print(6 / 4)
```

```
10
2
24
1.5
```

We also have

- % modulo/modulus
- // floor divide

```
print(6 % 4)
print(6 // 4)
```

```
2
1
```

Assignment

= is the assignment operator. Values on the LEFT are assigned values on the RIGHT.

```
x = 10
print(x)
```

```
10
```

In the code above, the value `10` is being assigned to the variable `x`. The variable `x` is then evaluated before it is passed into the `print` function.

Shorthand assignment

```
x += n  # is equivalent to x = x + n
x -= n  # is equivalent to x = x - n
x *= n  # is equivalent to x = x * n
x /= n  # is equivalent to n = x / n
```

```
x = 10
x += 2
```

```
print(x)

x = 10
x -= 2
print(x)

x = 10
x *= 2
print(x)

x = 10
x /= 2
print(x)
```

```
12
8
20
5.0
```

Equality

== is the equality operator. It compares the value on the LEFT to the value on the right.

```
print(10 == 5 + 5)
print(10 == 1 + 5)
```

```
True
False
```

Boolean operators

The logical boolean operators in Python are:

- `not` (flips the boolean value)
- `and`
- `or`

Truth table for `and`

and	True	False
True	True	False
False	False	False

```
def my_function():
    """This is a docstring."""
    pass

def next_hailstone_number(n):
    """
    Returns the hailstone number following the number `n`.
    This function assumes `n` is an integer. If `n` is not an␣
 ↪integer,
    the behaviour of this function is undefined.
    """
    if n % 2 == 0:
        return n // 2
    return 3 * n + 1
```

Week 06

Streams and files

Input and output streams

A stream is a flow of data – into or out of a program.

- Streams flowing into the program are input streams: keyboard, compact disk
- Streams flowing out the program are output streams: monitor, hard drive

Streams only exist while a program is running. After the program ends, the stream is closed. On the other hand, files can remain after programs end.

This is useful because:

- Files can be reused by different programs
- Convenient way to deal with large amounts of data
- Does not require user effort

File types

There are, generally, two types of files.

1. Binary files – less space used, read and edited by a program
2. Text files – human readable, can be edited by a human

There are no rules to what can be stored in a file.

File paths

Files are stored in your file system and can be referenced by a path.

A file path can be written in two ways:

1. Absolute path (from root directory, \)
2. Relative path (relative destination from current working directory)

File modes

When you open a file for reading (read mode), the program assumes the file exists and you have permission to read from the file.

When you open a file for writing (write mode), you always begin with an empty file.

- If file already exists, program will override
- If file does not exist, program will create it

To add to the end of an existing file, you will want to append (append mode).

Opening files in Python

The following code writes the string contents to a file called `sample.txt`. After running the below command in your terminal, you should be able to see a new file with the `ls` command.

```
%%bash
cat << EOF > sample.txt
Apple
Banana
Carrot
EOF
```

Reading from a file

We use the `open` function to open a file.

The `open` function takes a filepath as its first argument and a mode as the second argument. In this case, we want to read from the file, so we specify `"r"` as reading mode.

```
f = open("sample.txt", "r")
print(f)
f.close()
```

```
<_io.TextIOWrapper name='sample.txt' mode='r' encoding='UTF-8'>
```

Remember to `close` your files after you finish working with them.

Why do we need to close files?

- It signals to the operating system that we are done working with the file.
- If we opened the file for writing, closing the file will flush the buffer and make sure our changes are saved on disk.
- Having too many files open at once (like having too many programs open at once) will slow down the computer.

There are two common ways to read from files: