

INFO1110 & COMP9001: Introduction to Programming

School of Information Technologies, University of Sydney



COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Lecture 2: Programming basics

What is a program?

A program is a set of instructions that were written, probably by a human, in such a way that can be converted to instructions that a computer can understand and then execute.

Wikipedia has this to say:

A computer program (also software, or just a program) is a sequence of instructions written to perform a specified task with a computer.

Your First Program

The *classic* program of all time is “HelloWorld”. In Python it looks like this:

```
1 print("Hello World!")
```

Wait, that was just some text.

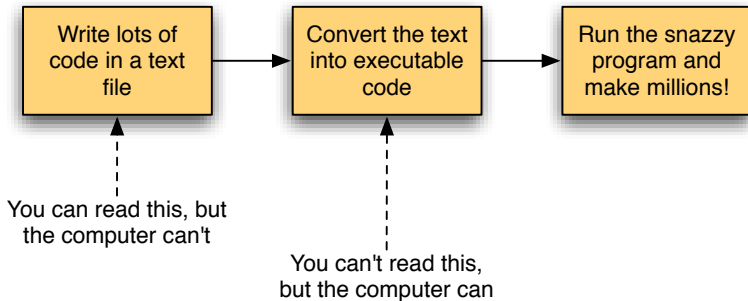
Ok, true.

When you write a program you really are writing a text file, or a set of text files, which you might even link with other programs.

The text file you write has to be in a special, very particular format, such that it obeys the *syntax* of the programming language you are using.

Once the text file is written, a *compiler* is used to process the file and turn it into something that a computer can run: an *executable*, a *program*, or an *application* or *app*.

Wait, that was just some text. (cont.)



There are many programming languages

...and no “best one”. We have chosen Python for this course as it is a major language used in industry, is cross-platform and has some nice features that make it very suitable for Rapid Application Development.

Here are some major languages:

C, C++, C#, BASIC, VisualBasic, Java, Python, Perl, Pascal, Haskell, Fortran, Cobol.

In old (1978) C it would be more like this:

```
1  int main() {  
2      printf("Hello, World!\n");  
3  }
```

in ISO-C++ it would be like this,

```
1  #include <iostream>  
2  
3  int main()  
4  {  
5      std::cout << "Hello World!" << std::endl;  
6  }
```

Languages (cont.)

in Perl like this:

```
1 print "Hello World!\n";
```

Javascript:

```
1 <!DOCTYPE HTML><html><body><script>  
2     alert( 'Hello World!' );  
3 </script></body></html>
```

What we have for all of these programs is something like the following:

```
1 <load things we need>      # "optional"  
2 <begin a method>           # explicit or implicit, "main"  
3 <print the string>         # definitely needed  
4 <end the printed line somehow>  
5 <end the method>           # as in line #2
```

Example program

This program will just print out a couple of messages.
Don't worry if you don't know what each line does; that's why you're here!

```
1 print("Boo!")  
2 print("Argh!")
```

Compiling and running

After making the text file, a.k.a. “the .py file”, we can compile and run **in one step**

On the command line (or “terminal”) you do this:

And *run* it like this:

```
~> python Surprise.py  
Boo!  
Argh!
```



When you type `python` and then the name of a program on the command-line, you’re invoking the Python interpreter to run your program. The Python Interpreter is itself a program, which enables python programs to run on many different computer platforms.^[1]

^[1]there are multiple versions of Python and this makes the statement untrue

Python has layers

```
1 print("Hello World!")
```

is similar to

```
import stdio
```

```
stdio.writeln("Hello World")
```

which is similar to

```
import sys
```

```
sys.stdout.write("Hello World")  
sys.stdout.write('\n')  
sys.stdout.flush()
```

- Which lines are instructing the Python Interpreter to print the right message when the program is run?
- What are the other lines doing?

Anatomy of the Hello World program

Comments begin with a #

```
# we need to include other code for this program  
import stdio
```

```
# we use the following line to print to screen  
stdio.writeln("Hello World!")
```

This is called an object.
Everything is an object!

This is our data

This is called a method. Methods can be in any order, and they can "call" each other to perform smaller actions. This one is calling a method called "writeln" to print the data we provide

In order for the compiler to turn your code into a working program, the code has to obey a lot of syntax rules. You will pick up many of these as you go along, but let's just list a few now too:

- Some words are *reserved* — e.g., `import`, `int`, `float`, `if`, `else`. You can't use these for variable names.
- Variable names can't begin with numbers.
- Blocks of code are delineated with braces { }
- Expressions are delineated with parentheses ()
- Array items are accessed with brackets []
- Control flow statements should end with colons :
- Strings are delimited by double quotes " " or single quotes ' '
- The code is only executed if it is indented correctly (ouch!)

It's very easy to get things wrong. Don't worry. Most things are fixable. **DIVE IN!**

The important bits in Hello, World

The main part that you have to worry about and understand is the line which does the real work. `stdio.` is a special variable, an *object* of a *class*, that has a *method* defined for it called `writeln`. It's the `writeln` method that actually prints a string of characters to the console.

`writeln` also puts a newline at the end of whatever it prints, so you don't have to add a newline character, `'\n'`.

You don't have to remember all this new terminology now

To *run* the program you type

```
> python HelloWorld.py  
Hello World
```

where it's assumed that `HelloWorld.py` has no errors

Too easy...

Go write some code!

Making a Program Go: Compiling a program

In order to make a Python (or Java, C, C++) program actually do anything it has to be *compiled*. This is a process done by a *compiler* that converts your human-readable (Python) code into machine-readable *byte code* or *machine code*.

To *compile* the source code. You would type

```
> compiler_program    human_readable_source_code
```

The output is machine readable instructions and data.

To run the program, the computer is fed instructions and data

Making a Program Go: Compiling a program

You don't need to compile with Python! it happens automatically but that does not mean it is not happening!

```
> python -m py_compile HelloWorld.py
```

(followed by a Return) and that should produce no output, which means there aren't any errors detected by the compiler.

This makes a *.pyc file* for your program, which is where the bytecode is (all being well).

In this case the class file would be called `HelloWorld.class`, and would *not* be human-readable (nor can I represent it here on a slide).

```
> python HelloWorld.pyc  
Hello World!
```

Basic input and output

```
1 response = input()  
2 print(response)
```

```
> python HelloInput.py
```

What happens?

Basic input and output

```
1 print("Do you like quokkas? ")
2 response = input()
3 print(response)
```

```
> python HelloInput.py
Do you like quokkas?
i am not sure
i am not sure
```

Basic input and output

input (string) var = input("what is your name") .

var → Oscar . print(var) → Oscar .

|||||

```
1 hs29uf2 = input("Do you like quokkas? ")
2 print(hs29uf2)
```

Oscar .

```
> python HelloInput.py
Do you like quokkas? i am not sure
i am not sure
```