Introduction to Programming

Dr. John Stavrakakis

School of Computer Science, University of Sydney

Copyright Warning

Lecture 2: Programming basics

We may use code from the course text

In this course, you may find that we need other files and code to run our programs based on the course text book by Sedgewick. They are a necessary part of the course for us to build understanding of programming with Python.

See Resources in Ed:

Code stdio.py

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:

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

Wait, that was just some text.

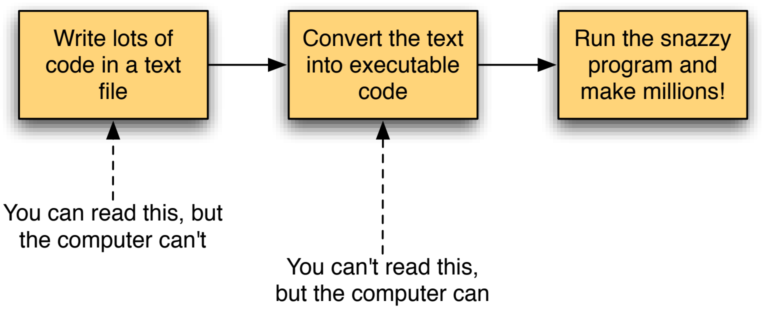
Ok, true.

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.)


Write lots of code in a text file
Convert the text into executable code
Run the snazzy program and make millions!
You can read this, but the computer can't
You can't read this, but the computer can

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.

Languages

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

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

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

```cpp
#include <iostream>

int main()
{
   std::cout << "Hello World!" << std::endl;
}
```

Languages (cont.)

in Perl like this:

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

Javascript:

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

In general...

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!

```python
print("Boo!")
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

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

is similar to

```
import stdio

stdio.writeln(“Hello World”)
```

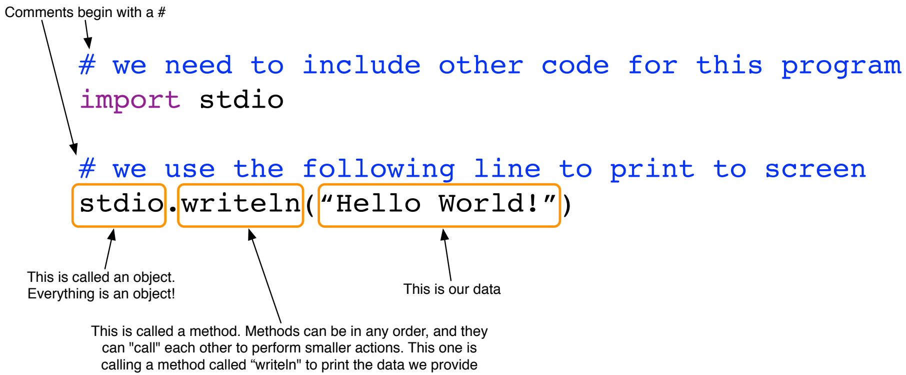
which is similar to

```
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

Python syntax

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.
- 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

Running it

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

```python
response = input()
print(response)
```

```
> python HelloInput.py
```

What happens?

Basic input and output

```python
print("Do you like quokkas? ")
response = input()
print(response)
```

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

Basic input and output

```python
user_answer = input("Do you like quokkas? ")
print(user_answer)
```

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

Testing Code

We will spend more time on this later, but for now, just try to remember that you *cannot* guarantee your code will work!

- Not the first time
- Not for every case
- Even if it compiles beautifully

So you need to run your program many times with different inputs and make sure it works. *Testing is really important.*

Syntax and Logic and Compiler

When you write programs you have to write them in such a way that they make sense to the compiler – there's a *syntax*, that is, a set of rules, that you must follow in order for this to be true.

We have the same thing in English: we can't just make up a sentence of random words and expect it to make sense.

randomizing it's of enough words bad order the just the.

Tguhoh aaltpeprny if you lvaee the fsrit and lsat ltetres in pacle it's ok

Syntax is not Logic

However even if we have the correct *syntax* we may still not make sense. This is a very famous syntactically correct yet meaningless sentence:

$$\boxed{\textit{Colorless green ideas sleep furiously}}$$

Noam Chomsky

In most cases your compiler will complain because of *syntax* errors, but it will also sometimes give you an error message if a variable is unused, or used before it’s been initialised, or if there is part of the code that is unreachable. These are *logic* errors. The compiler won’t be able to pick up on every logic error (else there would be no need for debugging!) but it will help you find simple logical errors. Pay attention to those compiler messages!