wellcome centre human genetics

Programming Concepts

October 2021

Ben Wright
Duncan Parkes

Who is this course for?

- People with no previous programming experience
- People who've 'learned by doing' as part of their work or studies but want a better grounding
- People who want a refresher after not programming for a while

What is programming?

- Creating a set of instructions for a computer to follow
- Three main challenges:
  - Getting it to do anything at all
  - Getting it to do what you told it do to
  - Getting it to do what you actually wanted

What makes computers tick?

- Computers use **machine code**
- When this code is **run**, it takes **input** and produces **output**
- Different languages are different ways of specifying instructions, all of which are turned into machine code when run

Compiled versus Interpreted

- **Compiled** languages require you to run a special tool called a compiler to turn the code you've written into machine code
  - Examples: C, C++, Fortran
- Compiling is done for a specific operating system (and sometimes processor architecture),
- If your code needs to run on different computers you may need to compile it for each of them.

Compiled versus Interpreted

- **Interpreted** languages perform this compilation behind the scenes whenever you run your code
  - Examples: Python, R, bash scripts
- Most programs in interpreted languages are **portable** - which means you can run them on any computer that has the right interpreter available

Compiled versus Interpreted

- Compiled languages can let you catch some errors before you try to run your code
- Compiled languages, usually, run faster (although this matters almost never)
- Interpreted languages are easier to experiment with (using a live interpreter)
- Interpreted languages are more portable

Running Code

- Anything you type into an interpreter is carried out immediately
  - e.g. in R, or the bash command line
- **Scripts** are simple text files that an interpreter treats the same way as typed text
  - e.g. Bash scripts, R scripts, most python code
- Scripts are important because they are also a record of what you did!

Running Code

- Compiled languages require you to run the relevant compiler on your code, then run the program you've created
- This extra step can be laborious when trying to fix bugs
- There exist '**Integrated Development Environments**' (IDEs) to make it easier
  - In particular, highlighting any compilation errors as you type

Pseudocode

- Pseudocode is halfway between describing the program in sentences and the code itself
- There's no formal definition of what it looks like
- It can be used as a way to work out the logical steps of your program before you try to write it

Pseudocode Example

- What does this do?

```
Set C to 0
For each value A in a list of values:
  If A is 0, set C to 0
  Otherwise, set C to C+1
Print C
```

Keywords

- Most languages have a set of **reserved keywords**
- These are special terms that have specific meaning in the language, and can't be used as anything else
  - Examples: if, for, import, return, elif, fi

Symbols

- Languages can also assign special meaning to particular symbols
- They can be used to:
  - Control program flow
  - Mark parts of the code as special
    - e.g. Double quotes (“) being used to indicate strings
  - Perform tasks like addition and subtraction - in this case they are called **operators**

Comments

- Text in your code that is ignored when it is run or compiled is called a **comment**
- Almost all languages allow you to add comments
- Use comments to explain not just what your code is doing, but ***why*** it is doing it.
- Comments are vital!

Datatypes

- Data is stored in **variables**, which you name
  - Naming rules are usually quite generous
- Variables have a **type**
  - e.g. an integer, a text string, an array
- Different types are often handled and/or stored differently by the language
  - You often need to be aware of this to avoid errors

Numeric Datatypes

- Numbers are generally divided into integers and floating point numbers
  - Integers are perfectly precise
  - Floating point numbers have a wider range of values and support decimals
- Some languages also have differently-sized variation on either, to store larger/more precise values
- R is a special case as it's designed to work with numeric data with high precision

Other Datatypes

- **Boolean**: a True or False value
  - Many languages also treat other datatypes as 'truthy', according to their own rules
- **Character**: a single letter or symbol
  - Examples: “A”, “a”, “@”, “4”
- **String**: a sequence of characters
- Other datatypes such as **objects**, and data structures discussed later

Datatype Declaration

- In some languages, you have to explicitly **declare** the datatype for each variable you create
  - Examples: C, Java
- In otheres, you do not
  - Examples: R, python

Datatype Casting

- Casting means converting one data type to another
  - e.g. turning the string “413” into the integer 413
  - e.g. turning the integer 413 into the floating point value 413.0
- Some languages will attempt to cast variables automatically as needed, others require you to explicitly cast your variables
- Converting to and from strings is more involved and usually uses a different method

Data Structures

- **Data structures** store multiple bits of data in a single place
- The type of structure determines the relationship between those bits of data
  - e.g. an array implies an ordering to the data
- In languages that use objects, these structures are usually themselves objects

Arrays / Vectors

- A linear sequence of data, all of the same type, of a known length, is called an **array**
- You can directly access elements of an array by their **index** - their position in the array
- R is a special case where nearly everything is a **vector** (i.e. an array), even if only of length 1

Maps / Dictionaries / Lists

- A map is similar to an array, only instead of each element having an index based on its position, each element has a **key**, which can be an arbitrary data value
- Elements of a map are usually unordered
- They go by different names in different languages - **associative arrays**, **dictionaries** (in python) and **lists** (in R)

Other Data Structures

- Linked lists
- Queues
- Binary trees
- Records / tuples / structs
- Sets
- Matrices

Code Blocks

- A code **block** is a set of lines of code
- It's not necessarily a formal feature of a language, but it's an important concept when writing code
- A code block is an organisational unit within your program

Conditional Blocks

- When you want your program do different things depending on the data
- It starts with a test ('**if**')
  - Followed by a block that's run if the test was true
- (Optionally) other tests ('**else if**' or '**elif**')
  - Followed by a block that's run if this is the first test to be true
- (Optionally) a block of code which is run if none of the tests were true ('**else**')

Conditional Blocks

```
if x > 10:                  IF statement
    m = 2                   First block
    over = true             |
elif x > 5:                 ELSE IF statement
    m = 1                   Second block
    over = false            |
else:                       ELSE statement
    m = 0                   Third block
    over = false            |
```

Loops

- Loops let you repeat a block of code
- Three kinds of loop:
  - You know how many times to repeat the block (for loop)
  - Test before each iteration (while loop)
  - Test at the end of each loop (until loop)
- Not all types of loop are available in all languages

Loops

```
x = 7
y = 0
while x > 0:          While loop
   y = y + x * x      Repeated block
   x = x - 1          |
```

Functions / Methods / Subroutines

- You can define a block of code once and use it in many places throughout your code - this is called a **function**
- Making use of a function is **calling** that function
- Writing functions means you don't have to repeat yourself and helps reduce errors
- Languages come with many functions built into them - it's worthwhile checking to see if a function already exists before writing your own

Functions / Methods / Subroutines

- Languages come with many functions built into them - it's always worthwhile checking to see if a function already exists to do what you want to do
- Functions are evaluated when they are encountered in your code, so you can use a function that returns a value directly in place of a variable, even when calling other functions

Functions / Methods / Subroutines

- Functions are evaluated when they are encountered in your code, so you can use a function that returns a value directly in place of a variable, even when calling other functions
- Functions have four main parts:
  - **Name**
  - **Arguments** (or parameters)
  - **Code**
  - (Optionally) **Return value**

Function Name and Arguments

- The name is how you call the function when you want to use it
- The arguments of a function are a list of variables the function will use
  - When the function is called, these arguments need to be provided
  - Some languages permit default values for arguments defined in the function

Function Code

- The code block inside the function is run whenever the function is called
- Often the function only uses variables passed to it as arguments - this makes it easier to re-use the function
- When using existing functions, you only care about what the function does, not how it does it

Function Return Values

- Some, but not all, functions return a value when you use them
- This can be the result of a calculation, but it can also be information about what the function did
  - e.g. whether an algorithm converged or not
- If a function returns a value, this can be stored as a variable or used directly by your code
  - Nesting functions makes for briefer code, but can be less readable

Function Side Effects

- If a function changes a variable outside itself, produces output or makes any other change outside itself, this is called a **side effect**.
- Despite the name, a side effect can be the purpose of the function
- Not realising that a function has side effects is a common source of coding errors

Functions

```
def triangular(x):
    y = x*(x+1)/2
    return(y)
```

Function name
Function argument
Code block
|
Return value

Scope

- Variables are only visible to certain parts of your code, according to the rules of the language
  - Variables defined inside code blocks are often invisible outside, them for example
- Where a variable can be seen is called its scope
- Many languages have ways of defining variable scope explicitly

Libraries / Modules / Packages

- Because the number of functions available as part of the language can be huge, they are often organised into **libraries**
- Typically a default library is always available to your code
- Other libraries must be **imported** before they are available to you
  - This just means adding a line of code to say you wish to do so

Errors

- By default, when a serious error occurs the programme stops - a **crash**
- If you are lucky, you will get an explanation of the error that says:
  - What the error was
  - Where in your code the error happened
- Interpreters typically detect these problems and halt your code without crashing themselves

Errors

- Silent errors, which don't stop the programme but lead to incorrect results, are far more serious and harder to detect
- Never take on faith that your code is working properly just because it didn't crash

Stack Traces

- When a programme crashes, often you will be given a something called a **stack trace**
- Because functions call other functions a stack trace is a nested list of which functions were running
- The last function is where the error was detected
  - The cause could be an error in that function
  - The cause could be an earlier function calling another with an incorrect argument

Best Practises and Style

- These are recommendations, usually specific to a language, to make your code 'good'
- Having code that works properly is only half the battle
- You need to make your code as easy to understand as possible
  - For anyone who needs to use your code later - which could even be you, months or years down the line

Where can I find out more?

- There are courses available through central IT:
  - https://skills.it.ox.ac.uk/programming-starting-to-think-like-a-programmer-activity
- TED-Ed has a beginner's programming strand:
  - https://www.youtube.com/watch?v=KFVdHDMcepw
- Rosetta Code provides side-by-side comparisons of the same code in different languages:
  - http://rosettacode.org/wiki/Rosetta_Code