Conditional statements

Introduction to Programming in Python

Rae Harbird, r.harbird@ucl.ac.uk

Software development (1/2)

Software development consists of:

- Establishing the **requirements**
- Creating a **design**
- **Implementing** the code, and
- **Testing** the implementation.

Software development (2/2)

Software **requirements** specify *what* a program must accomplish.

Software **design** specifies *how* the requirements will be accomplished. Software design involves the design of **algorithms**.

An algorithm is a step-by-step process for doing a task, often first written in **pseudocode**.

```
Program: guess random number


Generate a random number
While user has not guessed the number:
     Ask user to input number
     If the input == the random number
          print “Hoorah!”
          End the program
     Otherwise:
          Print “Try again!”
```

Figure 1: Psuedocode

Testing

Good programmers always leave plenty of time for testing. Testing should be *methodical*, in order to maximise the chance of finding errors.

Flow of Control

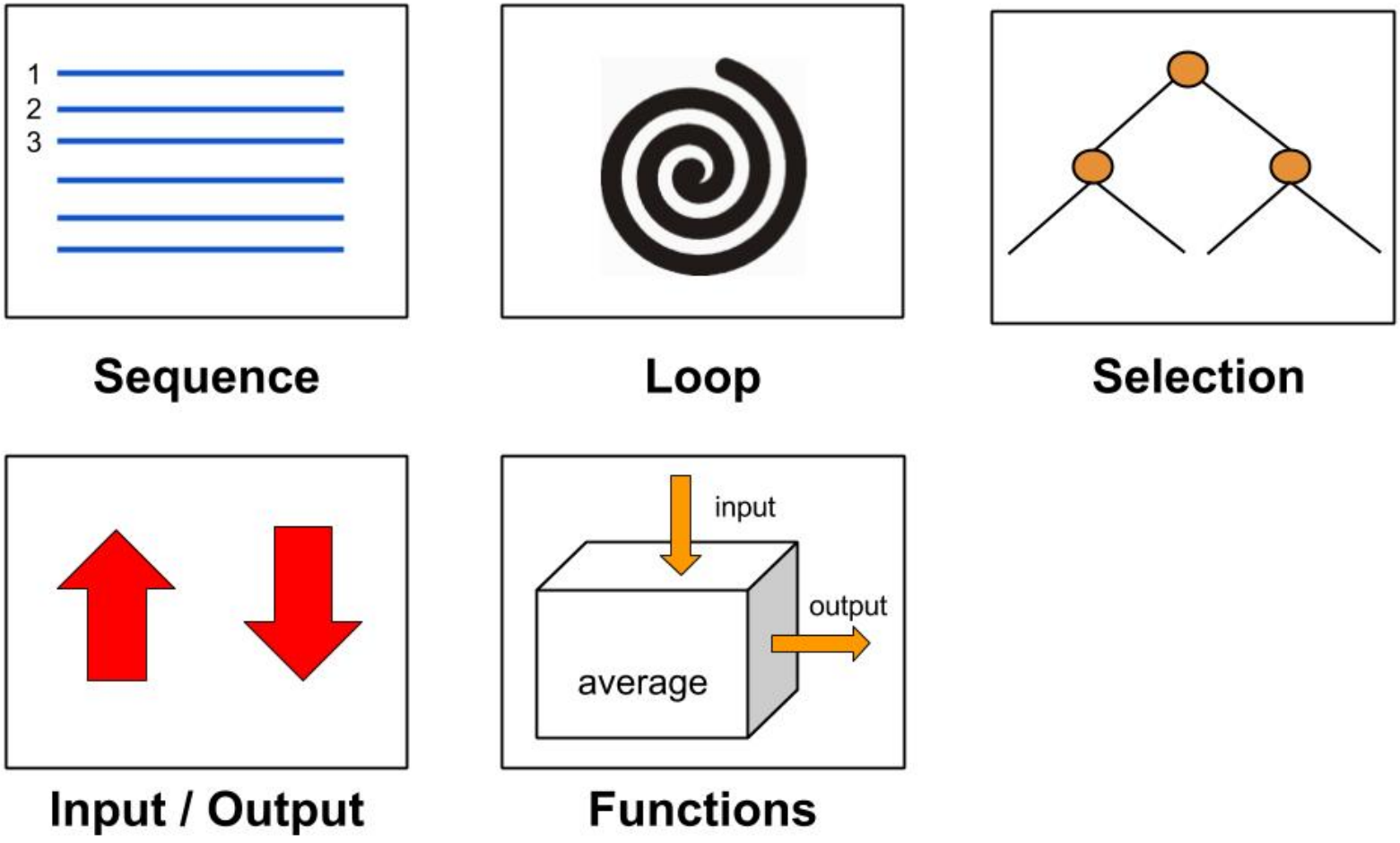

Figure 2: Control structures

Flow of Control - Explanation

The **flow of control** is the order that statements are executed in a program.

Unless otherwise specified, Python programs start with the first statement in the file and finish with the last (shown as sequence in the diagram).

This behaviour can be modified with selection statements, also called conditional statements, loops or repetition statements, input and output statements and functions. Any algorithm can be expressed in terms of these control structures.

Conditions

Conditional statements allow the program to **select** which statement will be executed next, based on the truth or falsity of a given **condition**.

The conditional statements in Python are the `if` statement, the `if-else` and the `if-elif-else` statements.

A Random Number Example

We will use variations of the following program in order to illustrate the use of conditional statements:

```
# Name: generate_random.py
# Description: This program generates a random number.

import random

MIN = 1
MAX = 5

def main():
    randomNumber = random.randint(MIN, MAX)
    print("A random number between " + str(MIN), end="")
    print(" and " + str(MAX) + ": " + str(randomNumber))

if __name__ == "__main__" :
    main()
```

Explanation

The program generates (and in this first variation outputs) a random number between `MIN` and `MAX` (including `MAX`). For example, the output might be:

```
A random number between 1 and 5: 4
```

The random module

The program `guess_one_random.py` includes the statement: `import random`. The `import` statement allows us to make use of code that other people have written, code is organised as libraries.

Take a look at the libraries available for your version of Python. Python's standard libraries include a module named **random** that contains many useful functions related to generating random values.

Technically the numbers generated by the library are called *pseudo-random numbers* because they are generated from mathematical functions and the system clock.

A call of `random.randint(1, 10)` returns a random integer from 1 through 10 inclusive.

Guessing With `if` Statements

The program `guess_one_random.py` uses two `if` statements to output two alternative messages. Each `if` statement has its own condition.

```
import random

MIN = 1
MAX = 5


def main():

    answer = random.randint(MIN, MAX)

    guess = int(input("\n\tGuess the number between " + str(MIN)
            + " and "  + str(MAX) + ": "))
    if guess == answer:
        print("\tCorrect - well done!")
    if guess != answer:
        print("\tNo, the answer was " + str(answer) + ".\n")
```

Note: the two conditions are exact "opposites".

Aside - using a main() function

It's good to structure your programs using a `main()` function. At the moment we won't go into the reasons for this. We'll explain the `main` function in more detail when we cover functions.

Guessing with an `if-else`

This code snippet from our program shows how the program could be modified to use a single if-else statement:

```
answer = random.randint(MIN, MAX)

guess = int(input("\\n\\tGuess the number between " + str(MIN)
                  + " and " + str(MAX) + ": "))
if guess == answer:
    print("\\tCorrect - well done!")
else:
    print("\\tNo, the answer was " + str(answer) + ".\\n")
```

Try it out. Note: there is no separate condition explicitly included in the "else" part.

Blocks and indentation

Python code is organised as blocks. We must indent all the statements to be executed if a condition is `True`, likewise, we must indent all the statements to be executed if the condition is `False`.

```
if totalSales > 100.0 :
↑  discount = totalSales * 0.05
|  totalSales = totalSales - discount
|  print("You received a discount of $%.2f" % discount)
else :
↑  diff = 100.0 - totalSales
|  if diff < 10.0 :
|  ↑  print("If you were to purchase our item of the day you can receive a 5% discount.")
|  else :
|  ↑  print("You need to spend $%.2f more to receive a 5% discount." % diff)
|  |  ↑
|  |  |
0  1  2  Indentation level
```

Figure 3: From: Python for Everyone

When editing code, it's best to use the `tab` key to indent your code. Your editor will probably allow you to choose the size of the `tab`, set it to 4 spaces. Don't mix spaces and tabs in the same program.

if-else statements in general

The general form of `if-else` statements is:

```
if condition :
    statement(s)
else :
    alternative statement(s)
```

If the condition evaluates to `True` then the indented statements(s) are executed, whereas if it is `False` the "alternative statement(s)" get executed.

The whole of the `else` part is optional.

Relational operators

Every `if` statement contains a condition. In many cases, the condition involves comparing two values. In Python, you use a *relational operator* to do this.

Python	Math Notation	Description
`>`	$>$	Greater than
`>=`	$\geq$	Greater than or equal to
`<`	$<$	Less than
`<=`	leq	Less then or equal to
`==`	$=$	Equal
`!=`	$\neq$	Not equal

Precedence and Relational Operators

Relational operators in have a lower precedence than arithmetic operators.

That means you can write arithmetic expressions on either side of the relational operator without using parentheses.

For example, in the expression:

```
floor - 1 < 13
```

both sides (floor - 1 and 13) of the $<$ operator are evaluated, and the results are compared.

Comparing strings

Strings can also be compared using Python's relational operators. For example, to test whether two strings are equal, use the == operator:

```
if name1 == name2 :
    print("The strings are identical.")
```

or to test if they are not equal, use the != operator:

```
if name1 != name2 :
    print("The strings are not identical.")
```

For two strings to be equal, they must be of the same length and contain the same sequence of characters. If even one character is different, the two strings will not be equal.

Strings and lexicographical ordering

Python uses *lexicographical ordering* to compare strings.

```
if "cat" > "bat":
    print("bat comes before cat")
else:
    print("cat comes before bat")
```

Output:

```
bat comes before cat
```

Test yourself

1. Give the opposite of the condition `floor > 13`
2. A programmer wrote the condition `floor = 13` to test whether the floor is 13. Fix the error.
3. Supply a condition in this `if` statement to test whether the user entered a Y:

```
response = input("Enter Y to quit.")
if . . . :
    print("Goodbye")
```

4. How do you test that a string variable: `response` is the empty string?

Common Error - Comparing floating point numbers

Floating-point numbers have only a limited precision, and calculations can introduce rounding errors.

For example, the following code multiplies the square root of 2 by itself. Ideally, we expect to get the answer 2:

```
from math import sqrt

r = sqrt(2.0)
if r * r == 2.0 :
   print("sqrt(2.0) squared is 2.0")
else :
   print("sqrt(2.0) squared is not 2.0 but", r * r)
```

Output:

```
sqrt(2.0) squared is not 2.0 but 2.0000000000000004
```

Solution - Comparing floating point numbers

To get around this, test whether the numbers are close enough. Mathematically, we would write:

$|x - y| < \epsilon$

Means that x and y are close enough if the difference between them is a very small number, ϵ.

It is common to set ϵ to $10^{-}14$ when comparing floating-point numbers:

```
from math import sqrt

EPSILON = 1E-14
r = sqrt(2.0)
if abs(r * r - 2.0) < EPSILON :
   print("sqrt(2.0) squared is approximately 2.0")
```

Nested if-else

`if-else` statements can be strung together as follows:

```
if boolean exp 1 :
    statement(s) 1
elif boolean exp 2 :
    statement(s) 2
elif boolean exp 3 :
    statement(s) 3
else :
    statement(s) 4
```

Python executes the first set of statements for which the corresponding boolean expression is `True`, and ignores the rest.

Why

For example, if `boolean exp 1` is false and both `boolean exp 2` and `boolean exp 3` are `true`, only `statement(s) 2` will be executed.

```
if boolean exp 1 :
    statement(s) 1
else :
    if boolean exp 2 :
        statement(s) 2
    else :
        if boolean exp 3 :
            statement(s) 3
        else :
            statement(s) 4
```

Guessing Two Numbers In Order

See program `guess_two_randoms.py` in the folder: `src`.

It's not perfect, we have to guess the numbers in a specific sequence. How can we improve this? Knowing how to use Boolean values will help, see the next video.