LEEDS
BECKETT
UNIVERSITY

Introduction to Programming

Week 4

Tony Jenkins

a.m.jenkins@leedsbeckett.ac.uk

OBJECTIVE

This week we will revise statements that control the flow of a program.

They will enable us to write DRY code (and not WET code).

Week 4
Pre-requisites

You should have viewed or read the lecture:

"Control Statements"

in the Part 3 folder on MyBeckett.

You should understand how Python stores and manipulates data - in variables.

You should be able to write simple programs that take input, do some processing, and display the results.

Working Code?
Code Smells

It is not enough for code to "just work".

Code must be efficient and maintainable.

A hint that the code is not efficient and maintainable is a "code smell" - something that doesn't feel quite right.

Duplicate (or near duplicate) code is a classic code smell - you will even find that your IDE may spot it and highlight it.

```
num1 = int(input('Enter ...'))
num2 = int(input('Enter ...'))
num3 = int(input('Enter ...'))

speed_1 = base_speed_1 * 1.6
speed_2 = base_speed_2 * 1.6
speed_3 = base_speed_3 * 1.6
speed_4 = base_speed_4 * 1.6
speed_5 = base_speed_5 * 1.6
speed_6 = base_speed_6 * 1.6
```

Flow of Control

Top Down?

To date, all our programs have executed top-down.

The statements are *interpreted* one at a time in strict sequence.

Now we want to be able to alter this *flow of control*. There are two possibilities:

1. Some code executes only if something is `True`.
2. Some code executes a number of times, until something becomes `True`.

("`True`" here could also be `False`, which is just `not True`!)

Flow of Control

Top Down?

All languages provide *control statements* to allow manipulation of the flow of control.

As usual, the details of what is on offer differ, but there will always be two general types:

- ➢ Conditional statements (usually "if", sometimes "switch").
- ➢ Loops (usually "while" and "for", sometimes "do", or even "repeat").

Flow of Control

Controlling What?

All languages provide *control statements* to allow manipulation of the flow of control.

Likewise, different languages offer different ways to show which statements are being controlled.

Many languages use curly braces. Some use special keywords.

Python uses indentation.

Conditionals
"if"

An "`if`" statement uses a Boolean expression to determine whether a statement should be executed.

Indented statements are controlled by the condition.

By convention, indentation is **four spaces**.

(Note that pressing TAB in PyCharm will usually insert four spaces, and that PyCharm will take care of most indentation for you.)

```
if name == 'Arthur':
    king = True

if swallow_speed > 10:
    print('Max Swallow Speed Found!')
```

Conditionals
"if"

An "if" statement uses a Boolean expression to determine whether a statement should be executed.

Additional possibilities can be handled using "elif" and "else" clauses.

The first condition found to be True is executed.

"else" serves as a "catch all". There does not have to be an "else".

```
if name == 'Arthur':
    king = True
elif name in ('Galahad', 'Robin'):
    knight = True

if swallow_speed > 10:
     print('Max Swallow Speed Found!')
elif swallow_speed > 5:
    print('Standard Swallow Detected!')
elif swallow_speed > 0:
    print('Slow Swallow Found!')
else:
    print('Data Error!')
```

Flow of Control
Loops

A programming language only really needs one form of loop. But for convenience there are usually two provided, and sometimes three.

The three possible loops are:

- ➢ Something must happen a known number of times.
- ➢ Something must happen while some condition is `True`.
- ➢ Something must happen until something is `True`.

There is a subtle difference between the last two - the second may not happen at all, the third will always happen once.

Loops
"for"

A "`for`" loop is used when it is known before the loop starts how many times it will execute.

It *iterates* over a *sequence*: the `range` function will generate a sequence of integers if needed.

A list can also be used as a sequence, as can a string.

More formally, this is called a *determinate loop*.

```
for i in range(10):
    print('Ni!')

for k in ['Robin', 'Galahad', 'Gawain']:
    print('Sir', k)

for c in password:
    if c in '0123456789':
        digit_found = True
```

Loops
"while"

A "`while`" loop is used when the number of iterations cannot be known in advance.

The code inside the loop executes while some condition is `True`.

Usually, something inside the loop alters the condition (otherwise the loop could be infinite).

```
while answer != 'n':

    .
    .
    .
    answer = input('Again? (y/n)')
```

Loops
"while True"

A "`while`" loop is used when the number of iterations cannot be known.

A common Python idiom is to use an infinite loop (where the condition is always `True`), and `break` out of the loop explicitly on the termination condition.

Note that:

```
while True:
```

is the same as:

```
while 1:
```

because `True` is 1 in disguise.

```
while True:

    .
    .
    .
    answer = input('Again? (y/n)')
    if answer == 'n':
        break
```

Program
Exam Marks

A student has five marks on five exams, graded 0 to 100.

Write a program that reads these five marks, and outputs the highest, lowest, and mean.

We can now *refactor* this program to remove duplication.

We should also be able to ensure marks are between 0 and 100.

And we should be able to generalise it to handle any number of marks.

Constants
(Sort of)

Another code smell can be the appearance of literal values in multiple places in a program, specifically the same literal value.

In the code opposite the literal 10 appears twice.

If we wanted to alter this code to handle 12 marks, we would need to make two changes - that's a *code smell*.

```
for m in range(10):
    mark = int(input('Mark? '))
    total += mark

avg_mark = total / 10
```

Constants
(Sort of)

The fix is to use a constant - like a variable but *immutable* (the value can't change).

(Python doesn't really have constants. By convention we write their names in UPPER_SNAKE_CASE to indicate that the values should not be changed).

So now to change from 10 to 12, just change the constant!

In real life, the constant would go in some config file and would be read when the program starts.

```
NUMBER_OF_MARKS = 10

for m in range(NUMBER_OF_MARKS):
    mark = int(input('Mark? '))
    total += mark

avg_mark = total / NUMBER_OF_MARKS
```

Program
Exam Marks

A student has five marks on five exams, graded 0 to 100.

Write a program that reads these five marks, and outputs the highest, lowest, and mean.

Use a constant, so that the number of marks can be changed easily.

For bonus credit:

- ➢ Validate the marks entered are in the correct range.
- ➢ Amend the program to work for *any number* of marks, terminated when the user just presses Enter.
- ➢ Find the *median* mark. (Hint: This is the sort of thing that just might be built-in somewhere.)
- ➢ Find the *standard deviation* of the marks.

Program
Password Checker

A certain University has a rule that all passwords used by its IT users must:

- ➢ Be between 8 and 12 character long.
- ➢ Contain at least one uppercase letter.
- ➢ Contain at least one lowercase letter.
- ➢ Contain at least one digit.
- ➢ Contain at least one "special character".

Write a program that accepts a string as input and displays whether or not it fits these rules and could be used as a password.

```
Password? Password1!
Password Check Result: Check OK.

Password? cheese
Password Check Result: Check Failed.

Password? bhH768jgh
Password Check Result: Check Failed.

Password? yellow9toaster%herring
Password Check Result: Check Failed.
```

NEXT

We continue to look at ways to write DRY code by examining the use of functions, for code reuse.

Thank you

LEEDS BECKETT UNIVERSITY