University of HUDDERSFIELD

CFS2160: Software Design and Development

04: Making Decisions

Choices, choices. If only.

Tony Jenkins
A.Jenkins@hud.ac.uk


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth result: '))
mark_5 = int (input ('Enter fifth result:  '))

total_marks = mark_1 + mark_2 + mark_3 + mark_4 + mark_5

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth
mark_5 = int (input ('Enter fifth

total_marks = mark_1 + mark_2 + ma

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

This program works, but what could go wrong?


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth
mark_5 = int (input ('Enter fifth

total_marks = mark_1 + mark_2 + ma

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

Also, how could we decide whether or not the student had passed?

Errors

Most possible errors are based on what the user enters:

- The student's name could be blank.
- A result could be entered out of range
  - An integer, but not 0 to 100.
- A value for the result could be entered that is not an integer.
  - A String, like "Pass".

```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth result: '))
mark_5 = int (input ('Enter fifth result:  '))

total_marks = mark_1 + mark_2 + mark_3 + mark_4 + mark_5

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

Errors

Most possible errors are based on what the user enters:

- The student's name could be blank.
- A result could be entered out of range
  - An integer, but not 0 to 100.
- A value for the result could be entered that is not an integer.
  - A String, like "Pass".

```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth result: '))
mark_5 =
total
avera
print
print                                                    ark))
```

There are different outcomes to these errors.

Some would stop the current program running, some would not.

Linear Code

So far, all the code we have written has been *linear*.

This means that *statements* are executed one at a time, from the top of the program to the bottom.

- Nothing is missed out.
- Usually, a program will execute until an error is encountered, or until there are no more statements.

This is all fine, but restrictive.

We need to be able to examine what's going on in the program, and make choices.

Booleans

Our choices will be based around the idea of a "Boolean" statement.

A Boolean statement (or expression) is something that is either "True" or "False".

Such an expression can be evaluated in a program, and different statements executed depending on the value.

Booleans

Our choices will be based around the idea of a "Boolean" statement.

A Boolean statement (or expression) is something that is either "True" or "False".

Such an expression can be evaluated in a program, and different statements executed depending on the value.

Note the important difference between == and =.

```
>>> eggs = 0
>>> eggs == 12
False
>>> eggs == 0
True

>>> eggs < 12
True
>>> eggs >= 0
True
```

Results

In our program, the user enters a result, that must be between 0 and 100.

We can now test this.

Results

In our program, the user enters a result, that must be between 0 and 100.

We can now test this.

```
>>> r = int (input ('Result: '))
Result: 50
>>> r >= 0 and r <= 100
True

>>> r = int (input ('Result: '))
Result: 130
>>> r >= 0 and r <= 100
False
```

Results

In our program, the user enters a result, that must be between 0 and 100.

We can now test this.

Note that the expression can be written in a bunch of different ways:

```
>>> r >= 0 and r <= 100
>>> r > -1 and r < 101
>>> not (r < 0 or r > 100)
```

```
>>> r = int (input ('Result: '))
Result: 50
>>> r >= 0 and r <= 100
True

>>> r = int (input ('Result: '))
Result: 130
>>> r >= 0 and r <= 100
False
```

Results

In our program, the user enters a result, that must be between 0 and 100.

We can now test this.

Note that the expression can be written in a bunch of different ways:

```
>>> r >= 0 and r <= 100
>>> r > -1 and r < 101
>>> not (r < 0 or r > 100)
```

```
>>> r = int (input ('Result: '))
Result: 50
>>> r >= 0 and r <= 100
True

>>> r = int (input ('Result: '))
R
```

A common "gotcha":

```
r >= 0 and <= 100
```

Conditional Statement

If we can express the condition, we can now write a *conditional statement*.

The keyword `if` introduces the expression, like so:

The colon (`:`) at the end of the line marks the end of the expression.

```
>>> r = int (input ('Result: '))
Result: 60

>>> if r >= 0 and r <= 100:
```

Conditional Statement

If we can express the condition, we can now write a *conditional statement*.

The keyword `if` introduces the expression, like so:

The colon (`:`) at the end of the line marks the end of the expression.

The next line is executed *only* if the statement evaluates to `True`.

```
>>> r = int (input ('Result: '))
Result: 60

>>> if r >= 0 and r <= 100:
```

Conditional Statement

If we can express the condition, we can now write a *conditional statement*.

The keyword `if` introduces the expression, like so:

The colon (`:`) at the end of the line marks the end of the expression.

The next line is executed *only* if the statement evaluates to `True`.

Which, here, it does.

```
>>> r = int (input ('Result: '))
Result: 60

>>> if r >= 0 and r <= 100:
...     print ('Result valid!')
...
Result valid!
```

Conditional Statement

If we can express the condition, we can now write a *conditional statement*.

The keyword `if` introduces the expression, like so:

The colon (`:`) at the end of the line marks the end of the expression.

The next line is executed *only* if the statement evaluates to `True`.

Which, here, it does.

```
>>> r = int (input ('Result: '))
Result: 60

>>> if r >= 0 and r <= 100:
...     print ('Result valid!')
...
R
```

The statement controlled by the conditional is *indented*.

Conditional Statement

If we can express the condition, we can now write a *conditional statement*.

The keyword `if` introduces the expression, like so:

The colon (`:`) at the end of the line marks the end of the expression.

The next line is executed *only* if the statement evaluates to `True`.

Which, here, it does.

```
>>> r = int (input ('Result: '))
Result: 60

>>> if r >= 0 and r <= 100:
...     print ('Result valid!')
...
R
```

The statement controlled by the conditional is *indented*.

There can be many statements, all indented to the same level.

Conditional Statement

If we can express the condition, we can now write a *conditional statement*.

The keyword `if` introduces the expression, like so:

The colon (`:`) at the end of the line marks the end of the expression.

The next line is executed *only* if the statement evaluates to `True`.

Which, here, it does.

```
>>> r = int (input ('Result: '))
Result: 60

>>> if r >= 0 and r <= 100:
...     print ('Result valid!')
...
R
```

The statement controlled by the conditional is *indented*.

It is customary to indent by **four** spaces (do not use tabs!).

Simple Choices

So now we can extend the program with an additional rule:

"The student has passed if their overall average result is greater than (or equal to) 50."

This involves adding a simple conditional to the end of the program.

Simple Choices


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth result: '))
mark_5 = int (input ('Enter fifth result:  '))

total_marks = mark_1 + mark_2 + mark_3 + mark_4 + mark_5

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))

if average_mark >= 50.0:
    print ('Congratulations! ' + name + ' has passed!')
```

More Choices

We can now find out if the student has passed:

```
if average_mark >= 50.0:
    print ('Congratulations! ' + name + ' has passed!')
```

But if they fail, we print nothing at all.

More Choices

We can now find out if the student has passed:

```
if average_mark >= 50.0:
    print ('Congratulations! ' + name + ' has passed!')
```

But if they fail, we print nothing at all.

So we need to *add* some code that will be executed if the Boolean expression is `False`.

More Choices

If the student has not passed, they *must* have failed, so:

```
if average_mark >= 50.0:
    print ('Congratulations! ' + name + ' has passed!')
else:
    print ('Sadly, ' + name + ' has failed. What a pity.')
```

Now, *either* the first `print` *or* the second will be executed, but never both.

More Choices

If the student has not passed, they *must* have failed, so:

```
if average_mark >= 50.0:
    print ('Congratulations! ' + name + ' has passed!')
else:
    print ('Sadly, ' + name + ' ha
```

Now, *either* the first `print` *or* the second will be

The `else` is indented to the same level as the `if`.

Other indentation shows which statement is in which *block*.

Indentation

Indentation is *crucial* in Python.

It marks out what statements are controlled by what other statements.

- Which statements are inside the `if`.
- (If you prefer), where the `if` statement block ends.
- Which statements are inside the `else`.

PyCharm will intelligently suggest indentation.

Indentation

Indentation is *crucial* in Python.

It marks out what statements are controlled by what other statements.

- Which statements are inside the `if`.
- (If you prefer), where the `if` statement
- Which statements are inside the `else`

PyCharm will intelligently suggest indentation.

The number of spaces in the indent is not important.

By convention, we use **FOUR** spaces.

Indentation


```
if some_condition:
    # This statement is inside the if.
    # So is this one.
    # And this one.
else:
    # This statement is inside the else.
```

Indentation


```
if some_condition:
    # This statement is inside the if.
    # So is this one.
    # And this one.
else:
    # This statement is inside the else.

# This statement is back in the main flow of the program.
```

Indentation


```
if some_condition:
    # This statement is inside the if.
    # So is this one.
    # And this one.
else:
    # This statement is inside the else.

# This statement is back in the main flow of the program.

    # This statement is an error!
```

More Complex Choices

Sometimes there are more than two options.

- ➢ Think 'A' Level or GCSE grades.

A more realistic rule might be:

"The student has passed if their overall average result is greater than (or equal to) 50. In addition, the student is awarded a Distinction if their average result is greater than or equal to 70."

To do this, we just need to add an extra choice.

More Complex Choices


```
if average_mark >= 70.0:
    print ('Cool Beans! ' + name + ' has a Distinction!')
elif average_mark >= 50.0:
    print ('Congratulations! ' + name + ' has passed!')
else:
    print ('Sadly, ' + name + ' has failed. What a pity.')
```

Refactoring


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth result: '))
mark_5 = int (input ('Enter fifth result:  '))

total_marks = mark_1 + mark_2 + mark_3 + mark_4 + mark_5

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

Refactoring


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth
mark_5 = int (input ('Enter fifth

total_marks = mark_1 + mark_2 + ma

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

We have here a "Code Smell".

This code works, but look at the duplication.

Refactoring


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth
mark_5 = int (input ('Enter fifth

total_marks = mark_1 + mark_2 + ma

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

We have here a "Code Smell".

Let's *refactor* to arrive at a better solution.

Refactoring


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth
mark_5 = int (input ('Enter fifth

total_marks = mark_1 + mark_2 + ma

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

We have here a "Code Smell".

We need to see how to *repeat* statements.

Refactoring


```
name = input ('Enter the student\'s name: ')

mark_1 = int (input ('Enter first result:  '))
mark_2 = int (input ('Enter second result: '))
mark_3 = int (input ('Enter third result:  '))
mark_4 = int (input ('Enter fourth
mark_5 = int (input ('Enter fifth

total_marks = mark_1 + mark_2 + ma

average_mark = total_marks / 5

print ()
print ('Final Mark for ' + name + ' is ' + str (average_mark))
```

We have here a "Code Smell".

We also need to consider the best *data structure* to use.

Refactoring

Issues with this program include (but are not limited to):

- We have five almost identical prompts.
  - It would be good to replace them with one prompt, and have the code *repeat.*
- We have five integer variables, used for almost the same thing.
  - We could replace them with a single data structure, like a Tuple.
- The results entered could be out of range.
  - We can detect this, but what do we do about it?

So now we go on to see how to fix these things.

Refactoring

Issues with this program include (but are not limited to):

- We have five almost identical prompts.
  - It would be good to replace them with one prompt, and have the code *repeat.*
- We have five integer variables, used for almost the same thing
  - We could replace them with a single data structur
- The results entered could be out of range.
  - We can detect this, but what do we do about it?

So now we go on to see how to fix these things.

The message here is that it's not enough for a program to work, it must work efficiently.

PyCharm Demo and Question Time

Jobs

By next week, you should:

- Have read up to the start of Unit 4 in the book.
  - Specifically, looked at different ways to combine Boolean expressions.
- Have skimmed (at least) Unit 4 in the book.
- Practiced.
- Attempted the "Capstone Project" at the end of Unit 3 in the book.
- Practiced some more.
- Got answers to any questions you might have.


python
powered