ECMM455 Python Worksheet 7: Conditional statements

Prof Hywel Williams

September 12, 2019

1 Aims

- Practice using conditional statements
- Practice using while loops

2 Conditional statements

Conditional statements execute different instructions depending whether a given condition (logical expression) is *True* or *False*. There are three basic forms of conditional statement in Python: *if, if-else, if-elif*. These can be combined to make more complicated decision structures.

2.1 Indentation

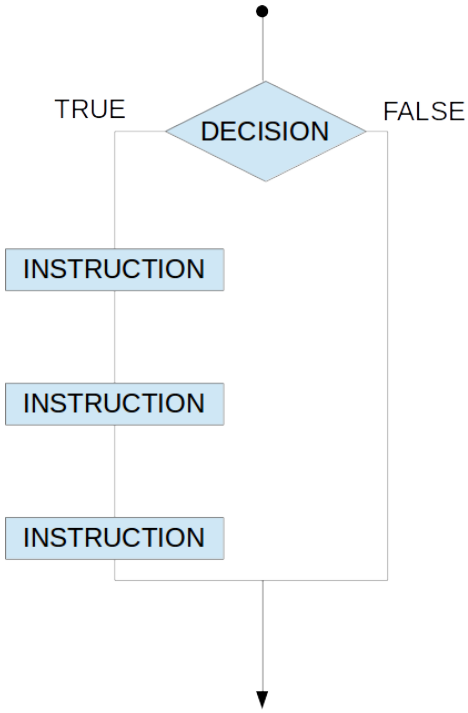
Python really cares about code indentation and whitespace. Indentation is used to identify the instruction set associated with a control structure, e.g. *if* statements use indentation to identify the associated instruction set, as we will see below. When indentation returns to original setting the instruction set ends.

DO NOT USE TABS TO INDENT! Different interpreters treat tabs differently. Use spaces instead (e.g. four spaces for each indentation level).

BE CONSISTENT! Whatever indentation method you use, the most important thing is to be consistent - problems with indentation arise when you start to mix styles...

2.2 The *if* statement

The *if* statement allows a set of commands to be executed if a condition evaluates *True*, but not otherwise. The flow of control looks like this:



The syntax for an *if* statement looks like this:

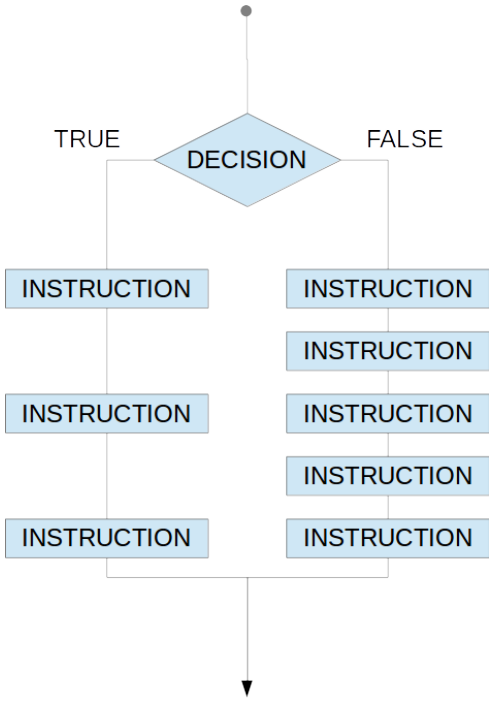
```
if <logical expression>:
  <instruction set>
```

A simple example looks like this:

```
if x>2:
  print("x is greater than 2")
```

The if-else statement

The *if-else* structure allows a set of commands to be executed if a condition evaluates *True* and a different set of commands if the condition evaluates *False*. The flow of control looks like this:



The syntax for an *if-else* statement looks like this:

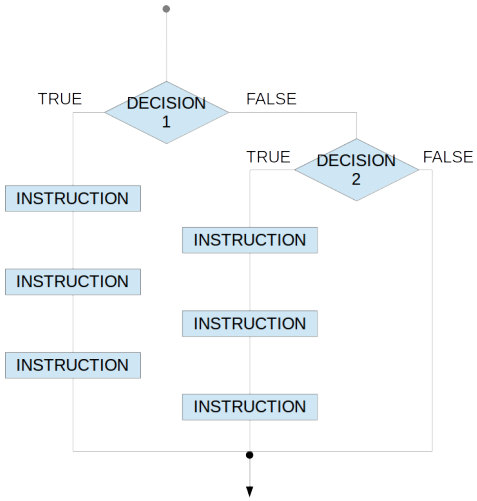
```
if <logical expression>:
  <instruction set A>
else:
  <instruction set B>
```

A simple example looks like this:

```python
if x>2:
  print("x is greater than 2")
else:
  print("x is not greater than 2")
```

2.3 The if-elif statement

The *if-elif* structure allows a set of commands to be executed if a first condition evaluates *True* and a different set of commands if the first condition evaluates *False*, but second condition evaluates *True*. The flow of control looks like this:



The syntax for an *if-elif* statement looks like this:

```
if <expression1>:
  <instruction set A>
elif <expression2>:
  <instruction set B>
```

A simple example looks like this:

```python
if x>2:
  print("x is greater than 2")
elif y<5:
  print("x is not greater than 2 and y is less than 5")
```

2.4 Exercises

1. Write a program that asks the user for a password and tells them if they get it right or not.
2. Write a program that asks the user for two numbers and then displays a message saying which number is larger.
3. Write a program that identifies an animal based on whether it has fur and can fly:
   - fur & can fly → bat
   - fur & can't fly → dog
   - no fur & can fly → pterodactyl
   - no fur & can't fly → human
4. Write a program that prompts the user for a percentage score between 0 and 100. If the score is out of this range, display an error message. If the score is between 0 and 100, display a grade using the following table:

Score	Grade
>=70	I
>=60	IIi
>=50	IIii
>=40	III
<40	Fail

5. Write a program that prompts the user for a temperature in either Fahrenheit or Celsius, distinguished by a C or F at the end of the string they enter. The program should convert the temperature in Fahrenheit (f) to the temperature in Celsius (c) using the formula $c = \frac{5}{9}(f - 32)$, or Celsius to Fahrenheit using the formula $f = \frac{9}{5}c + 32$, depending on what the user entered, then display the answer. For example, if the user entered "25C" the program should display "77F", or if the user entered "41F" then program should display "5C".

3 while loops

3.1 Conditional loops

A *while* loop is a conditional loop - it will iterate as long as some condition is evaluated as True. Syntax is given below:

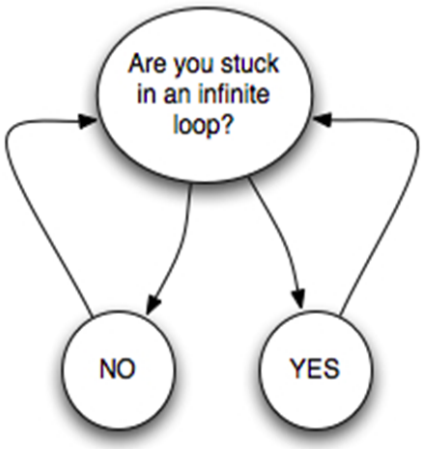
```
while <logical expression>:
  <instruction set>
```

A simple example is given below:

```
x=1
while x<10:
  print("x now has value: " + str(x))
  x+=1
```

3.2 Infinite loops

While loops can be very useful, but handle with care... if the condition never becomes False, code execution will enter an infinite loop!



For example, look at the code below. Can you see what is wrong?

```
x=1
while x<10:
  print("x now has value: " + str(x))
  x-=1
```

3.3 Exercises

1. Write a program that uses a while loop to count down from 10 to 1.
2. Write a program that asks the user to enter a word beginning with "A". The program should keep asking until a suitable word is entered.
3. Use a while loop to calculate how many years it will take to double an investment with an interest rate of 6%.
4. Write a program that will prompt the user for a list of positive numbers, one at a time. The program should keep a running total of the sum of all numbers entered. If the user enters a negative number, the running total (not including the negative number) should be displayed and the program should terminate.
5. Write a program to display an infinitely long seven-times-table, without ever stopping (i.e. it should count up in sevens, towards infinity, displaying each number in the sequence). Remember that you can terminate an infinite loop by pressing CTRL-C.