MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Programming in python™ for Business Analytics (BMAN73701)

Week 1/ Lecture 2 – Conditionals and Loops

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Last week

- General info about the course
- Whetting your appetite
- Setting up your Integrated Analysis Environment
- Python essentials
  - Your first program + Using Python as a Calculator
  - Numeric data types — int, float, complex
  - String operations — print(), input(), concatenation (, vs + notation)
  - Type conversion — int(), str(), float()
  - Variables — case sensitive, cannot start with numbers
  - In-place operators — x = x +2 equivalent to x += 2

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

A couple of things before we start

- Numeric data types: int, float, complex
- Text data types: string

Today:

- Sequence types (e.g. list, tuple, range)
- mapping types (dictionary)
- Boolean data type: True and False
- NoneType: Type for the None object, which is an object that indicates no value (this is different than zero). Occurs e.g. if one of your variables is not set, hence it has the value of None. More about this later.

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Agenda

- Commenting code
- Built-in types
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- Control flow statement
  - for and while loops
  - if, else, elif statements
  - break and continue statements
- Comparison and Boolean operations

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Agenda

- **Commenting code**
- Built-in types
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- Control flow statement
  - for and while loops
  - if, else, elif statements
  - break and continue statements
- Comparison and Boolean operations

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Commenting code properly

We receive this code (e.g. for your coursework)…

```
1 user_name = input("Enter your name: ")
2 height = input("Enter your height in feet: ")
3 height = float(height)
4 height = height/0.032808
5 height = round(height,3)
6 height = str(height)
7 print(user_name + " is " + height + " cm tall")
```

Can you follow this code easily?

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Commenting code properly

Then we receive this code…

- We can see:
  - Name of file and student
  - Comments on what lines do
- Goal: have 1 comment for every 1-4 lines of code
- Comment what code is doing and any design choices

```
1  #Richard Allmendinger
2  #feet_To_cm.py
3
4  #Obtain user name and height in feet
5  user_name = input("Enter your name: ")
6  height = input("Enter your height in feet: ")
7
8  #Convert feet into cm (1ft = 30.48cm)
9  #and round to 3 decimal places
10 height = round(float(height)*30.48,3)
11
12 print(user_name,"is",height,"cm tall")
```

Readability of code will be considered in coursework assessment

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Ten commandments of programming*

1. Think first about what you want to do before writing any code.
2. Break your problem into small independent blocks. Think about what precisely you want each block to do.
3. Prepare the project, i.e., create a folder and start with a fresh .py file.
4. Work "Inside to Outside": Identify the "core functionality" in each block, make it work first, and only then refine and extend.
5. If necessary, use the internet to get help (google "Python how to ...").
6. However, do not just copy-and-paste. Understand the main idea and write your own code!
7. Verification: Test your code line by line as you write it! Your computer always does exactly what you tell it to do. So if the result is different from the expectation defined in Step 2, it's your fault!
8. Read Python error messages. If you just added one line and the code "doesn't work" anymore, the problem is likely related to this line.
9. Always put meaningful comments into your code as you write it. Undocumented code is useless!
10. Be proud of what you achieved.

*Taken from http://nbviewer.jupyter.org/url/personalpages.manchester.ac.uk/staff/stefan.guettel/py/02a-loops_conditionals.ipynb#Ten-commandments-of-programming

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Agenda

- Commenting code
- **Built-in types**
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- Control flow statement
  - while and for loops
  - if, else, elif statements
  - break and continue statements
- Comparison and Boolean operations

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types

Done last week...

- The principal built-in types are numerics, sequences, mappings, classes, instances and exceptions
- Sequence types are used to group together other values
- Three basic sequence types: list, tuple, range
- There are also additional sequence types tailored for processing of binary data and text strings → later today

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - list

- Lists are the most versatile sequence type
- Typically used to store collections of **homogenous items** (but items can also be of different types)
- Initialization of a list with n items:

  list_name = [item_0, item_1,…, item_n-1]

  (initialize an empty list using **list_name = list()** or **list_name = []**)
- Lists are **mutable**, i.e. items can be appended, removed, changed, etc after assignment.
- If you are going to iterate over the items then use a list (vs tuple, see later)

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - list

Accessing items in a list

- All built-in sequence types can be *indexed* and *sliced*
- While *indexing* is used to obtain **individual elements**, *slicing* allows you to obtain a **set**
- Indic… to st…
- No… en…
- Slice indices have … defaults; an omitted first index defaults to zero, an omitted second index defaults to the size of the string being sliced.

Note: All slice operations return a new list containing the requested elements

```
>>> even = [2,4,6,8]
>>> even
[2, 4, 6, 8]
>>> even[0]
2
>>> even[-1]
8
>>> even[0:2]
[2, 4]
>>> even[2:4]
[6, 8]
>>> even[:2]
[2, 4]
>>> even[1:]
[4, 6, 8]
>>> even[-2:]
[6, 8]
```

Built-in types: Sequence types - list

Extending a list

```
4 li =[2,4]
5
6 #Append an item with value 6
7 li.append(6)
8 print(li)
9
10 #Append an item with value 6
11 li += [6]
12 print(li)
13
14 #Insert an item with value 99 at the 2nd position
15 li.insert(1,99)
16 print(li)
17
18 #Append items with value 8 and 10
19 li.append([8,10])
20 print(li)
21
22 #Append items with value 12 and 14
23 li.extend([12,14])
24 print(li)
```

```
[2, 4, 6]
[2, 4, 6, 6]
[2, 99, 4, 6, 6]
[2, 99, 4, 6, 6, [8, 10]]
[2, 99, 4, 6, 6, [8, 10], 12, 14]
```

Built-in types: Sequence types - list

Replacing items in a list

```
5 li =[2,4,6,8]
6
7 #Change first item to 1
8 li[0] = 1
9 print(li)
10
11 #Change 2nd to 4th items to 2, 3, 4
12 li[1:] = [2,3,4]
13 print(li)
```

```
[1, 4, 6, 8]
[1, 2, 3, 4]
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - list

Removing items from a list

```
5  li =[2,4,6,8,10]
6
7  #Remove 2nd item in list
8  del(li[1])
9  print(li)
10
11 #Remove 2nd and 3th item in list
12 del(li[1:3])
13 print(li)
14
15 #Remove the item with value 10
16 li.remove(10)
17 print(li)
```

```
[2, 6, 8, 10]
[2, 10]
[2]
```

Built-in types: Sequence types - list

Sorting a list

```
4 li =[5,-2,3,1,7]
5
6 #Sort list in ascending order
7 li.sort()
8 print(li)
9
10 #Sort list in descending order
11 li.sort(reverse=True)
12 print(li)
13
14 #Reverse list
15 li.reverse()
16 print(li)
```

```
[-2, 1, 3, 5, 7]
[7, 5, 3, 1, -2]
[-2, 1, 3, 5, 7]
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - list

Other useful operations for a list

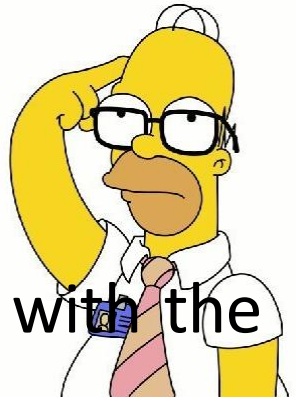
```
4 li =[1,5,-2,3,1,7]
5
6 #Check if a particular element is in the list
7 print(3 in li)
8 print(10 in li)
9
10 #Count the number of times an element is in the list
11 print(li.count(1), li.count(10))
12
13 #Obtain length of list
14 print(len(li))
15
16 #Retrieve min and max values of a list
17 print(min(li),max(li))
18
19 #Retrieve position of an item
20 print(li.index(5),li.index(1))
21 print(li.index(100))
```

```
True
False
2 0
6
-2 7
1 0
```

```
ValueError: 100 is not in list
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Quiz time



1. Fill in the blank to create a list called squares with the elements 1,4,9,16, and 25

>>> ____ [1,4,9__,25_

>>> squares = [1,4,9,16,25]

2. What is the output of the code >>> squares[-3:]?

a) [1,4,9]

b) [9,16,25]

b) is correct

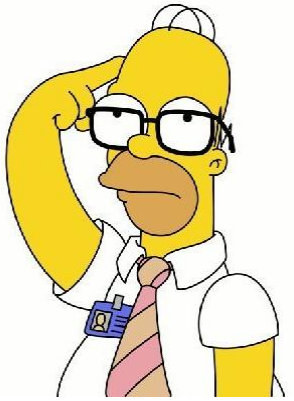
c) 9,16,25

3. What is the output of the code >>> squares[:]?

The output is [1,4,9,16,25]

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Quiz time



3.b) What is the output of the code

```
4 li =[5,4,2,0,9]
5
6 li[1:3] *= 2
7 print(li)
```

```
[5, 4, 2, 4, 2, 0, 9]
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - tuple

- Tuples are **immutable**, i.e. items are fixed after assignment
- Used for collection of heterogeneous items, e.g. a person's details broken into (name, age, city)
- Generally used where order and position is meaningful and consistent
- Initialization of a tuple with n items:

  tuple_name = (item_0, item_1,…, item_n-1)

  (initialize an empty tuple using **tuple_name = ()** or **tuple_name = tuple()**)

Built-in types: Sequence types - tuple

```
>>> person1 = ('Richard A.', 25, 'Melbourne')
>>> person1
('Richard A.', 25, 'Melbourne')
>>> person1[1]
25
>>> person1[1:]
(25, 'Melbourne')
>>> person2 = ('Manuel L.', 35, 'Madrid')
>>> personAll = person1 + person2
>>> personAll
('Richard A.', 25, 'Melbourne', 'Manuel L.', 35, 'Madrid')
>>> len(personAll)
6
>>> 'Anna B.' in personAll
False
```

These operations are the same for lists

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - tuple

Adding items to a tuple

```
1 tup = (1,2,3)
2 print(tup)
3
4 #use (x,) to add one element, x, to a tuple
5 to_add = (4,)
6 tup += to_add
7 print(tup)
8
9 #a list can be transformed into a tuple
10 to_add = [5,6,7,8]
11 ttt = tuple(to_add)
12 tup += ttt
13 print(tup)
```

```
(1, 2, 3)
(1, 2, 3, 4)
(1, 2, 3, 4, 5, 6, 7, 8)
```

lists

```
4 li =[2,4]
5
6 #Append an item with
7 li.append(6)
8 print(li)
9
10 #Append an item with
11 li += [6]
12 print(li)
13
14 #Insert an item with
15 li.insert(1,99)
16 print(li)
17
18 #Append items with v
19 li.append([8,10])
20 print(li)
21
22 #Append items with v
23 li.extend([12,14])
24 print(li)
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - tuple

Replacing items in a tuple

```
1 t = ('275', '54000', '0.0', '5000.0', '0.0')
2 print(t)
3
4 lst = list(t)
5 print(lst)
6
7 lst[0] = '300'
8 print(lst)
9
10 t = tuple(lst)
11 print(t)
```

```
5 li =[2,4,6,8]
6
7 #Change first item t
8 li[0] = 1
9 print(li)
10
11 #Change 2nd to 4th i
12 li[1:] = [2,3,4]
13 print(li)
```

lists

```
('275', '54000', '0.0', '5000.0', '0.0')
['275', '54000', '0.0', '5000.0', '0.0']
['300', '54000', '0.0', '5000.0', '0.0']
('300', '54000', '0.0', '5000.0', '0.0')
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - range

- The range type represents an **immutable** sequence of numbers
- The range type is commonly used for looping a specific number of times in for loops.
- Initialization a range object with a sequence of numbers starting at *start* and finishing at *stop* with a step size of *step:* **range_name = range(start, stop, step)**
- The advantage of the range type over a regular list or tuple is that a range object will always take the same (small) amount of memory, no matter the size of the range it represents.

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Sequence types - range

```
4 #Create a range element from
5 #0 to 20 with step size 2
6 range_t = range(0,20,2)
7
8 #Print range variable
9 print(range_t)
10 print(list(range_t))
11
12 #Check if certain value is in range
13 print(10 in range_t)
14 print(11 in range_t)
15
16 #Obtain index of certain value
17 print(range_t.index(10))
18 print(range_t.index(11))
19
20 #Retrieve element(s) at a certain position
21 print(range_t[5])
22 print(range_t[:3])
23 print(range_t[-1])
```

```
range(0, 20, 2)
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
True
```

```
Yes
No
```

```
5
ValueError: 11 is not in range
```

```
10
range(0, 6, 2)
18
```

Built-in types: Sequence types - range

```
>>> list(range(0,10,1))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(0,30,5))
[0, 5, 10, 15, 20, 25]
>>> list(range(0,10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> list(range(0,10,3))
[0, 3, 6, 9]
>>> list(range(0,-10,-1))
[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
>>> list(range(0))
[]
```

range_name = `range` (start, stop, step)
- default values are start = 0, step = 1

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Text sequence types - str

- Textual data in Python is handled with str objects, or strings
- Strings are **immutable** sequences of Unicode code points
- Initialization of a string (both ‘ and “ can be used):

  string_name = ‘text’ or **string_name = str(‘text’)**

- We can apply the same operations to strings as we do for other immutable sequence types, such as indexing, slicing, etc. (see next slide for examples)
- Strings have also many additional methods (overview of these can be found at https://docs.python.org/3/library/stdtypes.html#str)

Built-in types: Text sequence types - str

```
>>> word = 'Python'
>>> word[0]
'P'
>>> word[-2]
'o'
>>> word[0:2]
'Py'
>>> word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> 'J'+word[1:]
'Jython'
>>> word[:2] + 'py'
'Pypy'
```

P	y	t	h	o	n
0	1	2	3	4	5	6
-6	-5	-4	-3	-2	-1	

Built-in types: Text sequence types - str

```
1 numbers = [1,2,3]
2 word = 'hello'
3 word_list = list(word)
4 numbers += word_list
5 print(numbers)
```

What is the output of this code?

```
[1, 2, 3, 'h', 'e', 'l', 'l', 'o']
```

Why?

A string, let's call it *x*, is a (text-)sequence, i.e. if it is an input in list(x) or tuple(x), then the items in the list will be split

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Common sequence operations

- These operations are supported by most sequence types, both **mutable** (e.g. lists) and **immutable** (e.g. tuple)

Operation	Result
`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`
`s + t`	the concatenation of *s* and *t*
`s * n` or `n * s`	equivalent to adding *s* to itself *n* times
`s[i]`	*i*th item of *s*, origin 0
`s[i:j]`	slice of *s* from *i* to *j*
`s[i:j:k]`	slice of *s* from *i* to *j* with step *k*
`len(s)`	length of *s*
`min(s)`	smallest item of *s*
`max(s)`	largest item of *s*
`s.index(x[, i[, j]])`	index of the first occurrence of *x* in *s* (at or after index *i* and before index *j*)
`s.count(x)`	total number of occurrences of *x* in *s*

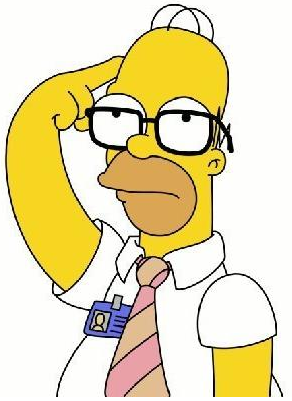
https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: **Mutable** sequence operations

Operation	Result
`s[i] = x`	item *i* of *s* is replaced by *x*
`s[i:j] = t`	slice of *s* from *i* to *j* is replaced by the contents of the iterable *t*
`del s[i:j]`	same as `s[i:j] = []`
`s[i:j:k] = t`	the elements of `s[i:j:k]` are replaced by those of *t*
`del s[i:j:k]`	removes the elements of `s[i:j:k]` from the list
`s.append(x)`	appends *x* to the end of the sequence (same as `s[len(s):len(s)] = [x]`)
`s.clear()`	removes all items from `s` (same as `del s[:]`)
`s.copy()`	creates a shallow copy of `s` (same as `s[:]`)
`s.extend(t)` or `s += t`	extends *s* with the contents of *t* (for the most part the same as `s[len(s):len(s)] = t`)
`s *= n`	updates *s* with its contents repeated *n* times
`s.insert(i, x)`	inserts *x* into *s* at the index given by *i* (same as `s[i:i] = [x]`)
`s.pop([i])`	retrieves the item at *i* and also removes it from *s*
`s.remove(x)`	remove the first item from *s* where `s[i] == x`
`s.reverse()`	reverses the items of *s* in place

Quiz time



4. Provide code that does the following:

   a) Create a list called *letters* containing the items ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘f’, and ‘g’.

   b) Return the letter ‘a’ from the list (*letters*).

   c) Check if the letter ‘t’ is in the list.

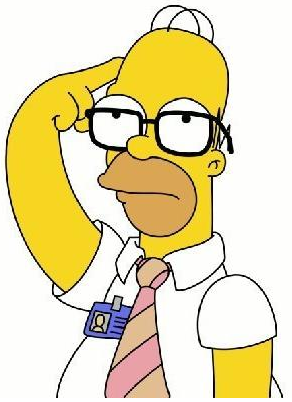
   d) Return the length the list.

   e) Append the letter ‘h’ to the list.

   f) Replace the items ‘c’, ‘d’, ‘e’, by ‘C’, ‘D’, ‘E’.

   g) Remove the items ‘C’, ‘D’, ‘E’.

```
word = 'abcdefg'

#alternatively, letters could be defined
#using letters = ['a','c','c','d','e','f','g']
letters = list(word)
print(letters)

#return letter 'a'
print(letters[0])

#check if letter 't' is in the list
print('t' in letters)

#length of list
print(len(letters))

#append 'h' to list
letters.append('h')
print(letters)

#replace the items ‘c’, ‘d’, ‘e’, by ‘C’, ‘D’, ‘E
#first find out the index of the items ‘c’, ‘d’, ‘e’,
#and then do the replacement
index_c = letters.index('c')
letters[index_c] = 'C'

index_d = letters.index('d')
letters[index_d] = 'D'

index_e = letters.index('e')
letters[index_e] = 'E'
print(letters)

#remove the items ‘C’, ‘D’, ‘E’.
letters.remove('C')
letters.remove('D')
letters.remove('E')
print(letters)
```



```
['a', 'b', 'c', 'd', 'e', 'f', 'g']
a
False
7
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']
['a', 'b', 'C', 'D', 'E', 'f', 'g', 'h']
['a', 'b', 'f', 'g', 'h']
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Agenda

- Commenting code
- **Built-in types**
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- Control flow statement
  - for and while loops
  - if, else, elif statements
  - break and continue statements
- Comparison and Boolean operations

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Built-in types: Mapping types - dict

- Currently one standard mapping type in Python, the *dictionary*
- Dictionaries are indexed by *keys*, which can be any immutable type, e.g. strings, numbers or tuples.
- The *values* of a dictionary can be of any type.
- Think of a dictionary as an unordered set of *key: value* pairs, with the requirement that the keys are unique (within a dictionary).
- Initialization of a dictionary (more than two options):

dict_name = {key_1 : value_1,…, key_n : value_n} or
dict_name = dict([(key_1,value_1),…,(key_n,value_n)])

Built-in types: Mapping types - dict

- Dictionaries have many useful operations, see https://docs.python.org/3/library/stdtypes.html#dict for an overview

The same dictionary can have key:value pairs of different types.

```
>>> tel
>>> tel
{'manuel
>>> tel[
1425
>>> tel['anna'] = 9283
>>> tel
{'anna': 9283, 'manuel': 4769, 'richard': 1425}
>>> del tel['manuel']
>>> tel
{'anna': 9283, 'richard': 1425}
>>> 'anna' in tel
True
>>> tel.keys()
dict_keys(['anna', 'richard'])
>>> list(tel.keys())
['anna', 'richard']
```

```
>>> d = {"one": 1, "two": 2, "three": 3, "four": 4}
>>> d
{'one': 1, 'two': 2, 'three': 3, 'four': 4}
>>> list(d)
['one', 'two', 'three', 'four']
>>> list(d.values())
[1, 2, 3, 4]
>>> d["one"] = 42
>>> d
{'one': 42, 'two': 2, 'three': 3, 'four': 4}
>>> del d["two"]
>>> d["two"] = None
>>> d
{'one': 42, 'three': 3, 'four': 4, 'two': None}
```

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> a == b == c == d == e
True
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Agenda

- Commenting code
- Built-in types
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- **Control flow statement**
  - for and while loops
  - if, else, elif statements
  - break and continue statements
- Comparison and Boolean operations

Control flow statements - for

- The for statement is used to iterate over the elements of a sequence (e.g. a string, tuple or list) or other iterable object (typically *used if we know in advance* how may iterations should be done)

for **item** in *sequence*:

    indented statements to repeat; may use **item**

forLoopExample_week2.py

```
1 #Iterate through list of names
2
3 names = ['Richard', 'Manuel', 'Anna', 'Ella']
4
5 for n in names:
6     print(n)
```

```
Richard
Manuel
Anna
Ella
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Control flow statements - for

Some more examples …

```
1 #for statement example
2 for i in range(10):
3     print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```

What do we need to change in the code to get a
0 to 9

Why do we get this output?

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Control flow statements - for

Some more examples …

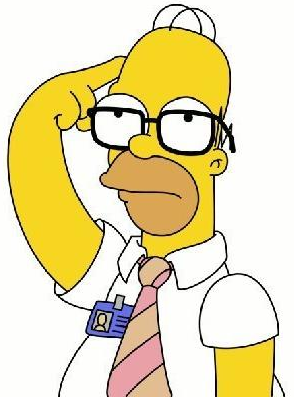
```
#A simple repeat loop

for i in range(10):
    print('Hello')
```

```
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
```

```
n = int(input('Enter the number of times to repeat: '))
for i in range(n):
    print('This is repetitious!')
```

```
Enter the number of times to repeat: 3
This is repetitious!
This is repetitious!
This is repetitious!
```

Quiz time



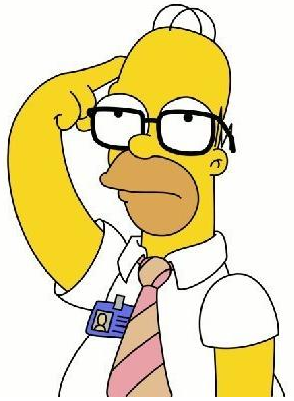
5a. Write a program that inputs an integer n, then inputs n numbers, $a_1,\ldots,a_n$ and prints $S = a_1 + \ldots + a_n$

How would we do it manually?

1. Input n.
2. Initialize S to 0 (meaning give S its initial value 0).
3. Then input a number, say x.
4. Add x to S, i.e., $S = S + x$.
5. Repeat steps 3 and 4 a total of n times.
6. Print the result S.

Since computers prefer to know in advance that we wish to loop through some of the code, we shall rearrange the above (i.e., put item 5 where the loop starts)

Quiz time



5a. Write a program that inputs an integer n, then inputs n numbers, $a_1,\ldots,a_n$ and prints $S = a_1 + \ldots + a_n$

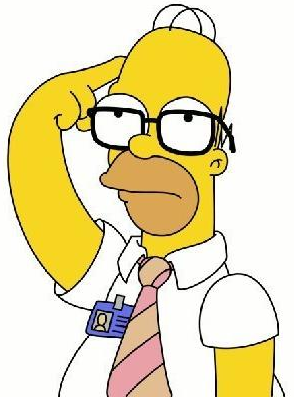
How would we do it manually?

1. Input n.
2. Initialize S to 0 (meaning give S its initial value 0).
3. Repeat the following n times:
   i. Input a number, say x.
   ii. Add x to S, i.e., $S = S + x$.
4. Print the result S.

Since computers prefer to know in advance that we wish to loop through some of the code, we shall rearrange the above (i.e., put the item 5 where the loop starts)

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Quiz time



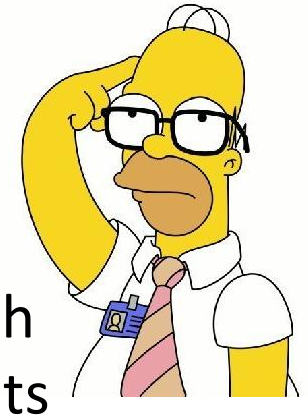
5a. Write a program that inputs an integer n, then inputs n numbers, $a_1,\ldots,a_n$ and prints $S = a_1 + \ldots + a_n$

```
1 n = int(input("How many numbers do you want to add up? "))
2 s = 0
3 for i in range(n):
4     x = float(input("Input a_" + str(i+1) + ": "))
5     s += x
6 print(s)
```

1. Input n.
2. Initialize S to 0 (meaning give S its initial value 0).
3. Repeat the following n times:
   i. Input a number, say x.
   ii. Add x to S, i.e., $S = S + x$.
4. Print the result S.

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Quiz time



5b. Write a program that first creates a list called *object* with the items 'red', 'orange', 'yellow', and 'green', and then prints out the name of each item alongside the item count, i.e.

red 1

orange 2

yellow 3

green 4

```
 8 objects = ['red','orange','yellow','green']
 9
10 count = 0
11 for item in objects:
12     print(item,count+1)
13     count+=1

15 print()
16 for count, item in enumerate(objects):
17     print(item,count+1)

19 print()
20 for count in range(len(objects)):
21     print(objects[count],count+1)
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Control flow statements - while

- The while statement is used for repeated execution as long as an expression is true
- Often used *if we do not know in advance* how may repetitions to be done)

```
while condition:
    indented statement block
```

```
for item in sequence:
    Indented statements to repeat; may use item
```

for vs while

for **item** in *sequence*:

    indented statements to repeat;
    may use **item**

while *condition*:

    indented statement block

```
1 n = int(input("n = "))
2 for i in range(n):
3     print(i)
```

```
1 n = int(input("n = "))
2 i = 0
3 while i < n:
4     print(i)
5     i += 1
```

For i in (0,1,2,3,…,n-1), print i

While i is less than n, print i and increase it by one

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Control flow statements - while

When is the condition checked?

while *condition*:

    intended statement block

```
i = 0
while i < 2:
    print("Before the increment:", i)
    i += 1
    print("After the increment:", i)
```

What is the output of this code?

```
Before the increment: 0
After the increment: 1
Before the increment: 1
After the increment: 2
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Quiz time

To do at home

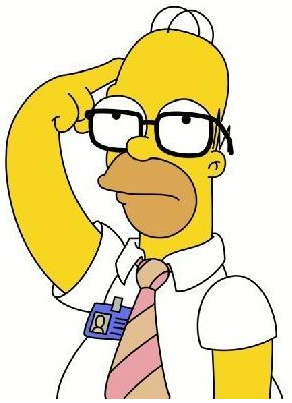
6a. Write a program that increases the temperature (starting from 15 degrees) iteratively by 1 degree until a certain temperature (25 degrees) is reached. Print out each new temperature and also a line indicating that the program has terminated.

temperature_week2.py

```
1  #Increase temperature until a
2  #certain heat is reached
3
4  temp = 15
5
6  while temp < 25:
7      print(temp)
8      temp = temp + 1
9
10 print('The water is warm enough!')
```

Line	temp	Comment
4	15	
6		15 < 25 is true, do loop
7		prints 15
8	16	15 + 1 is 16, loop back
6		16 < 25 is true, do loop
7		prints 16
8	17	16 + 1 is 17, loop back
…		
8	24	23 + 1 is 24, loop back
6		24 < 25 is true, do loop
7		prints 24
8	25	24 + 1 is 25, loop back
6		25 < 25 is false, skip loop
10		Prints that the water is warm enough

Quiz time



6b. What is the output of this code?

```
1 a = 0
2 b = 1
3 c = a + b
4
5 while c < 10:
6     print(c)
7     a = b
8     b = c
9     c = a + b
```

```
1
2
3
5
8
```

***Fibonacci* series:**

$F_n = F_{n-1} + F_{n-2}$ with seed values $F_0 = 0$ and $F_1 = 1$

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Agenda

- Commenting code
- Built-in types
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- **Control flow statement**
  - for and while loops
  - if, else, elif statements
  - break and continue statements
- Comparison and Boolean operations

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Control flow statements - if

- The if statement is used for conditional execution

if *condition*:

    indented statement block if condition is true

else:

    indented statement block if condition is false

```
1 #Graduation example
2
3 credits = int(input("Input number of credits: "))
4
5 if credits > 120:
6     print("Congratulations! You are eligible for graduation!")
7 else:
8     remain = 120 - credits
9     print("Sorry, you need "+str(remain)+" credits more to graduate.")
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Control flow statements - if

Convert a numerical grade to a letter grade, ‘A’, ‘B’, ‘C’, ‘D’ or ‘F’, where the cutoffs for ‘A’, ‘B’, ‘C’, and ‘D’ are 90, 80, 70, and 60 respectively.

```
1  #Score example
2  score = int(input('What score do you have? '))
3  if score >= 90:
4      letter = 'A'
5  else:   # grade must be B, C, D or F
6      if score >= 80:
7          letter = 'B'
8      else:  # grade must be C, D or F
9          if score >= 70:
10             letter = 'C'
11         else:    # grade must D or F
12             if score >= 60:
13                 letter = 'D'
14             else:
15                 letter = 'F'
16
17 print('Your grade is '+letter)
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Control flow statements - if- elif

```
#Score example
score = int(input('What score do you have? '))
if score >= 90:
    letter = 'A'
else:   # grade must be B, C, D or F
    if score >= 80:
        letter = 'B'
    else:  # grade must be C, D or F
        if score >= 70:
            letter = 'C'
        else:    # grade must D or F
            if score >= 60:
                letter = 'D'
            else:
                letter = 'F'

print('Your grade is '+letter)
```

```
#Score example
score = int(input('What score do you
if score >= 90:
    letter = 'A'
else:   # grade must be B, C, D
    if score >= 80:
        letter = 'B'
    else:  # grade must be C, D or F
        if score >= 70:
            letter = 'C'
        else:    # grade must D or F
            if score >= 60:
                letter = 'D'
            else:
```

if *condition1*:

    indented statement block if condition1 is true

elif *condition2*:

    indented statement block if condition2 is true

elif *condition3*:

    indented statement block if condition3 is true

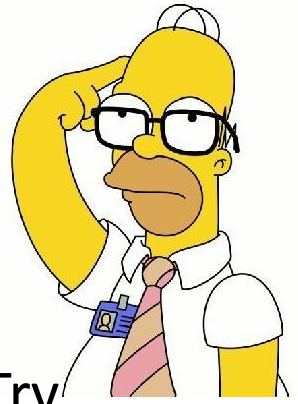
elif *condition4*:

    indented statement block if condition4 is true

else:

    indented statement block if each condition is false

Quiz time



7. Write a program that first asks the user for a number and then prints out if the number is positive, negative, or zero. Try using the commands if, elif and else in your code

```
1 #Determine sign of user-provided number
2 number = float(input('Please provide any number! '))
3 if number > 0:
4     print('Your number is positive')
5 elif number < 0:
6     print('Your number is negative')
7 else:
8     print('Your number is 0')
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Nested control flow statements

Suppose you have a list with the numbers [4, -2, 1, -8, 0, 4]. Print out only the positive numbers. **Use a loop and an if clause in your code.**

```
1 #Process positive numbers only in a list
2 numberList = [4, -2, 1, -8, 0, 4]
3 for number in numberList:
4         if number > 0:
5             print(number)
```

Nested control flow statements

Be careful when a (mutable) sequence is modified by a loop…

```
1 #Process positive numbers only in a list
2 numberList = [4, 2, -8, 3]
3
4 for number in numberList:
5         print(number)
6
7         if number > 0:
8             numberList.remove(number)
9
10        print(numberList)
```

```
4
[2, -8, 3]
-8
[2, -8, 3]
3
[2, -8]
```

numberList	index	number	number > 0
[4,2,-8,3]	0	4	True
[2,-8,3]	1	-8	False
[2,-8,3]	2	3	True
[2,-8]	3	for loop stops because no 3rd item in list	

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Nested control flow statements

Be careful when a (mutable) sequence is modified by a loop…

```
1 #Process positive numbers only in a list
2 numberList = [4, 2, -8, 3]
3
4 for number in numberList:
5         print(number)
6
7         if number > 0:
8             numberList.remove(number)
9
10        print(numberList)
```

```
4
[2, -8, 3]
-8
[2, -8, 3]
3
[2, -8]
```

Probably safest solution: Use while loop instead

```
3 numberList = [4,2,-8,-3]
4 counter=0
5 while counter < len(numberList):
6     print(numberList[counter])
7
8     if numberList[counter] > 0:
9         del numberList[counter]
10    else:
11        counter +=1
12
13    print(numberList)
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Nested control flow statements

Be careful when a (mutable) sequence is modified by a loop…

```
3 numberList = [4,2,-8,-3]
4 counter=0
5 while counter < len(numberList):
6     print(numberList[counter])
7
8     if numberList[counter] > 0:
9         del numberList[counter]
10    else:
11        counter +=1
12
13    print(numberList)
```

Probably safest solution: Use while loop instead

```
4
[2, -8, -3]
2
[-8, -3]
-8
[-8, -3]
-3
[-8, -3]
```

numberList	counter	numberList[counter] Line 6	Line 8 condition
[4,2,-8,3]	0	4	True
[2,-8,3]	0	2	True
[-8,3]	0	-8	False
[-8]	1	-3	False

Agenda

- Commenting code
- Built-in types
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- **Control flow statement**
  - for and while loops
  - if, else, elif statements
  - break and continue statements
- Comparison and Boolean operations

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Further control flow statements - break

- The break statement terminates the current loop and resumes execution at the next statement
- Most commonly used when some external condition is triggered requiring a hasty exit from a (*for* or *while*) loop.

```
for letter in 'Python':
   if letter == 'h':
      break
   print('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
```

```
for count in range(3):
   print(count)
   for letter in 'Python':
      if letter == 'h':
         break
      print('Current Letter :', letter)
   print(count)

print('Finished')
```

```
0
Current Letter : P
Current Letter : y
Current Letter : t
0
1
Current Letter : P
Current Letter : y
Current Letter : t
1
2
Current Letter : P
Current Letter : y
Current Letter : t
2
Finished
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Further control flow statements - continue

- The continue statement returns the control to the beginning of the loop

```
12 for letter in 'Python':
13     if letter == 'h':
14         continue
15     print('Current Letter :', letter)
```

```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

```
2 for count in range(3):
3     print(count)
4     for letter in 'Python':
5         if letter == 'h':
6             continue
7         print('Current Letter :', letter)
8     print(count)
9
10 print('Finished')
```

```
0
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
0
1
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
1
2
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
2
Finished
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Agenda

- Commenting code
- Built-in types
  - Sequence types (list, tuple, range, str)
  - Mapping types (dict)
- Control flow statement
  - while and for loops
  - if, else, elif statements
  - break and continue statements
- **Comparison and Boolean operations**

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Comparison operations

temperature_week2.py

```
#Increase temperature until
#certain heat is reached

temp = 15

while temp < 25:
    print(temp)
    temp = temp + 1

print('The water is warm enou
```

```
#Score example
score = int(input('What score do you have? '))
if score >= 90:
    letter = 'A'
else:   # grade must be B, C, D or F
    if score >= 80:
        letter = 'B'
    else:  # grade must be C, D or F
        if score >= 70:
            letter = 'C'
        else:    # grade must D or F
            if score >= 60:
                letter = 'D'
            else:
                letter = 'F'

print('Your grade is '+letter)
```

```
#Graduation example

credits = int(input("In

if credits > 120:
    print("Congratul
else:
    remain = 120 - c
    print("Sorry, yo
```

```
for letter in 'Python':
    if letter == 'h':
        break
    print('Current Letter :', letter)
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Comparison operations

- Eight comparison operations in Python
- **All have the same priority**
- Comparisons can be chained arbitrarily

Operation	Meaning
`<`	strictly less than
`<=`	less than or equal
`>`	strictly greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal
`is`	object identity
`is not`	negated object identity

Will become clearer next week

Supported for sequences only

`x in s`	`True` if an item of *s* is equal to *x*, else `False`
`x not in s`	`False` if an item of *s* is equal to *x*, else `True`

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Boolean operations

Boolean operations, **ordered by ascending priority** (all three **Boolean operators** have **a lower priority** than the **comparison operators** on previous slide):

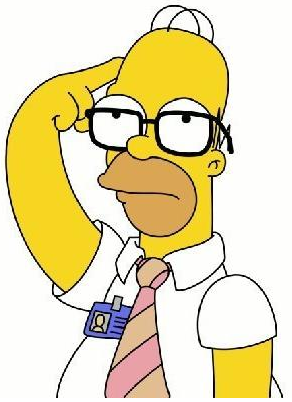
Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

```
1 #If stock is Google, the stock value below or
2 #equal to 50, and we have more than 100 stocks,
3 #then sell
4 stock_name = input('What stock do you have? ')
5 stock_value = float(input('Please provide the value per stock '))
6 stocks = int(input('How many stocks do you have? '))
7
8 if stock_name == "Google":
9     if stock_value <= 50:
10         if stocks > 100:
11             print('Sell stocks!')
12 else:
13     print('Do not sell stocks!')
```

Rewritten as

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Quiz time



8a. Which output will be created by the code and why?

```
1 a = 6
2 b = 7
3 c = 42
4 print(1, a == 6)
5 print(2, a == 7)
6 print(3, a == 6 and b == 7)
7 print(4, a == 7 and b == 7)
8 print(5, not a == 7 and b == 7)
9 print(6, a == 7 or b == 7)
10 print(7, a == 7 or b == 6)
11 print(8, not (a == 7 and b == 6))
12 print(9, not a == 7 and b == 6)
```

a)

```
1 True
2 False
3 True
4 False
5 True
6 True
7 False
8 True
9 False
```

b)

```
1 True
2 False
3 True
4 False
5 True
6 False
7 False
8 True
9 False
```

c)

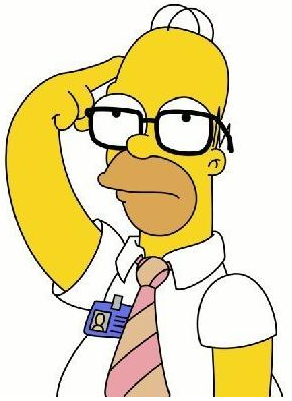
```
1 True
2 False
3 True
4 False
5 False
6 True
7 True
8 True
9 False
```

Operation	Result
`x or y`	if *x* is false, then *y*, else *x*
`x and y`	if *x* is false, then *x*, else *y*
`not x`	if *x* is false, then `True`, else `False`

Ordered in ascending order of priority

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

Quiz time



8b. Which output will be created by the code and why?

```
1 print("1 < 2:", 1 < 2)
2 print("1 < 1:", 1 < 1)
3 print("2 < 1:", 2 < 1)
4 print("1 < 2 < 3:", 1 < 2 < 3)
5 print("1 < 2 < 1:", 1 < 2 < 1)
6 print("1 < 2 > 3:", 1 < 2 > 3)
7 print("1 < 2 > 1:", 1 < 2 > 1)
8
9 a = 1
10 b = 2
11 c = 2
12 d = 0
13 print("1 < 2 == 2 >= 0:", a < b == c >= d)
14 print("1 < 2 == 0 >= 2:", a < b == d >= c)
15 print("1 < 2 >= 2 > 0: ", a < b >= c > d)
16 print("1 <= 2 >= 2 > 0:", a <= b >= c > d)
```

```
1 < 2: True
1 < 1: False
2 < 1: False
1 < 2 < 3: True
1 < 2 < 1: False
1 < 2 > 3: False
1 < 2 > 1: True
1 < 2 == 2 >= 0: True
1 < 2 == 0 >= 2: False
1 < 2 >= 2 > 0:  True
1 <= 2 >= 2 > 0: True
```

MANCHESTER 1824
The University of Manchester
Alliance Manchester Business School

A note on Boolean expressions

if *condition*:

    indented statement block if condition is true

- So far, `condition` had a Boolean value, e.g. score > 80 is either `True` or `False`.
- But most data structures/types can be used in a condition. Some general rules:
  - `None` (i.e. empty) and 0 (zero) evaluate to `False`.
  - Non-zero numbers evaluate to `True`.

```
0 is false.
17 is true.
x is false.
y is false.
1-1 is false.
True
False
True
```

```
1  x = 0
2  y = 1-1
3  if 0:
4      print("0 is true.")
5  else:
6      print("0 is false.")
7  if 17:
8      print("17 is true.")
9  else:
10     print("17 is false.")
11 if x:
12     print("x is true.")
13 else:
14     print("x is false.")
15 if y:
16     print("y is true.")
17 else:
18     print("y is false.")
19 if 1-1:
20     print("1-1 is true.")
21 else:
22     print("1-1 is false.")
23
24 print(bool(1))
25 print(bool(0))
26 print(bool(87))
```

Week 2/ Lecture 3 Preparation

- Prior to seminars
  - Revise lecture slides and do Quizes again
  - Have a go at Quizes and questions on BB

- Prior to lecture, have a look at
  - functions
  - modules
  - exceptions

- BB: Additional materials
  - Multiple choice questions
  - Videos to support you in understanding how to determine and interpret line equations