Exercises

In this notebook, we give you some unmarked exercises to try out. The answers are also provided, but try and work it out yourself first!

Question 1

Create a logical sequence for a number in variable `x`. Choosing your own values:

- If the number is too big, you should print the output "This number is too big"
- If the number is less than or equal to 0, you should print the output "This number is too small"
- Otherwise, you should print the output "This is in the correct range"

In [1]:

```
# YOUR CODE GOES HERE

x = 44
if x > 50:
    print("This number is too big")
elif x <= 0:
    print("The number is too small")
else:
    print("This is in the correct range")
```

```
This is in the correct range
```

Question 2

Put the code from Question 1 into a function called `is_number_valid`, which takes a parameter `x`, an integer to test against. The function should return `True` if the number is the correct value and `False` otherwise

In [ ]:

```
# YOUR CODE GOES HERE

def is_number_valid(x):
    x = 44
    if x > 50:
        print("This number is too big")
    elif x <= 0:
        print("The number is too small")
    else:
        print("This is in the correct range")
```

Question 3

Look at the documentation for the Python `range` (https://docs.python.org/3/library/stdtypes.html#range), use range to create a series of loops which do the following:

- Start at 0 up to and including 10 and print the output of each value

- Start at 30 and go up to and including 40, and print the square of each one (Hint: You can use `**` to square a value)
- Start at 100, and go down by 5 every time until you reach 50. Print half of the value for each iteration.

In [2]:

```
#  YOUR CODE HERE

for i in range(11):
    print(i)

for i in range(30,41):
    print(i ** 2)

for i in range(100,45, -5):
    print(i/2)
```

```
0
1
2
3
4
5
6
7
8
9
10
900
961
1024
1089
1156
1225
1296
1369
1444
1521
1600
50.0
47.5
45.0
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
```

Question 4

Expand the following class to include the following:

- A field called `tutees`, defaulting to an empty `dict`, when filled should have a key of a student's name (string) and a value of their mark (integet)
- Two fields `name`, and `subject` (both strings) to be added in the constructor (`__init__` method)

- A method `add_student` in the `Lecturer` class which takes a `Student` object as a parameter and adds them to the `tutees` dict.
- A method `update_mark` which takes parameters `student_name`, and `mark` and updates the value `mark` for the entry with the key `student_name`. If the student does not exist, the method should print a warning
- A method `print_tutees`, which iterates through the `Lecturer`'s tutees and prints their name and marks.

In [6]:

```
class Lecturer(object):
    name = ''
    subject = ''
    tutees = {}

    def __init__(self, name, subject):
        self.name = name
        self.subject = subject


    def add_student(self, name):
        # Could be improved by making sure the student does not exist first.
        # How would you do that?
        self.tutees[name] = ''

    def update_mark(self, name, mark):
        if not name in self.tutees:
            print('WARNING!  This student %s is not assigned to this lecturer %s' %
        self.tutees[name] = mark

    def print_tutees(self):
        for t in self.tutees:
            print('%s\t%d' % (t, self.tutees[t]))

students = ['Bob', 'Ivor', 'Rachel', 'Pat']
lec = Lecturer('Huw', 'Data Science')
for s in students:
    lec.add_student(s)

lec.update_mark('Bob', 54)
lec.update_mark('Frank', 39)
lec.update_mark('Pat', 66)
lec.update_mark('Rachel', 92)
lec.update_mark('Ivor', 24)

lec.print_tutees()
```

```
WARNING!  This student Frank is not assigned to this lecturer Huw
Ivor    24
Pat     66
Rachel  92
Frank   39
Bob     54
```

In [ ]:

```

```