Birds On A Wire

This week's project uses some loops, an array, some random numbers and some functions (which we create using `def`). These are all things that we've used before, so there shouldn't be any surprises.

Here's the idea. There's a long wire stretching between two telephone poles, and birds come down to try to land on the wire. I want to know how many birds can land on the wire before no more will fit.

This is a type of problem called *modelling*: we're going to describe the problem in a really simplified way so that we can represent it in a computer program. This is exactly the same idea (but MUCH simpler!) as using computers to predict the weather.

We're going to represent the bird as a blob with wings. Sitting on the wire, it will look like this: `----^O^----`. Here are the rules (in our model) of where a new bird can land:

- It cannot land on top of an existing bird
- It cannot overlap its wing with the body of an existing bird (`--^OO^-- is` not allowed)
- It can overlap its wing with the wing of an existing bird (`--^O^O^--` is allowed)

As usual, we will build up slowly to our final program. Let's get started.

As usual, start by opening the python shell (IDLE) and use `file->save as`. Navigate to the code club drive and set the name of your program. Mine is called neal_birds. Use your own name, so that everyone's program has a different name.

Type in this piece of code.

```
# Birds on a wire
#

from random import randint

# wire[0] to wire [131]
wire_length = 132
wire = ["x"]*wire_length

def clear_wire():
    for i in range(0,wire_length):
        wire[i] = "-"

def print_wire():
    for i in range(0,wire_length-1):
        print(wire[i], end="")


# test it..
print_wire()
clear_wire()
print_wire()
```

Now click `run->run module` (or press F5). When the program runs it should print out a line of "`x`" – representing the wire just after we created it, then a line of "`-`" representing the line after we cleared it. Change the size of your Python Shell window so that those lines fit without wrapping around onto multiple lines. If you get an error when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again.

The program uses a list, named `wire`, to represent the wire. Every element of the list (remember, this means `wire[0]`, `wire[1]`, `wire[2]` and so on up to `wire[131]`) starts up with the letter "x" in it (we could have chosen anything).

Next, we declare two functions, using `def` each time. The first one loops through each element of the list and (regardless of its current value) changes it to a "-". The second one loops through each element of the list and prints it out. Usually, each time that we use print(), the stuff that we print goes onto a new line. That's no use to us here: we need all of those "`-`" to be next to each other. The `end=""` in the print statement makes print end what it's doing without going on to a new line.

Finally, there are three lines to test the program. The first prints the wire to show what it looks like before it is cleared, the second one clears the wire and the third one shows what the wire looks like when it has been cleared.

If you are unclear about any of this, ask for help.

Unfortunately, that loop to print the wire is rather slow. When the program runs, you can see the line being printed, character by character (run it again if you didn't notice).

We can make a faster version of `print_wire`.

Modify your program to look like the one below. You don't need to type it all in; just the parts that have changed. The new code is shown in red. You also have to change the parts shown in green.

When you type the code in, your code will not be coloured like this!

```
# Birds on a wire
#

from random import randint

# wire[0] to wire [131]
wire_length = 132
wire = ["x"]*wire_length

def clear_wire():
    for i in range(0,wire_length):
        wire[i] = "-"

def print_wire():
    for i in range(0,wire_length-1):
        print(wire[i], end="")

def print_wire2():
    print("".join(wire))

# test it..
print_wire2()
clear_wire()
print_wire2()
```

As before, click `run->run module` (or press F5). When the program runs it should do just what it did before, but it should print much more quickly. As before, if you get an error when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again.

Next, we define (using `def` again) a function add_bird, which attempts to land a bird on the wire at a particular position.

Modify your program to look like the one below. You don't need to type it all in; just the parts that have changed. The new code is shown in red. Notice that the section starting "`# test it`" has been deleted; make sure you delete it from your program.

When you type the code in, your code will not be coloured like this!

```
# Birds on a wire
#

from random import randint

# wire[0] to wire [131]
wire_length = 132
wire = ["x"]*wire_length

def clear_wire():
    for i in range(0,wire_length):
        wire[i] = "-"

def print_wire():
    for i in range(0,wire_length-1):
        print(wire[i], end="")

def print_wire2():
    print("".join(wire))

# Try to fit a bird at the position 'where'. If it's possible, return True
# and add the bird. If it's not possible, return False
def add_bird(where):
    if wire[where] == "-":
        if wire[where-1] == "-" or wire[where-1] == "^":
            if wire[where+1] == "-" or wire[where-1] == "^":
                # yes!
                wire[where] = "O"
                wire[where-1] = "^"
                wire[where+1] = "^"
                return True
    return False
```

As before, click `run->run module` (or press F5). When the program runs it should do.. *nothing* (did you delete the testing stuff from the bottom? If not, delete it from your program and run it again). Most importantly, *it should not report any errors*. If you do get any errors when you try to run it, compare the code above carefully with what you typed. Correct any mistakes and try again.

After you have run the program, click in the Python Shell window and type in some commands by hand to test it out:

```
clear_wire()
add_bird(30)
print_wire2()
add_bird(31)
add_bird(32)
add_bird(33)
add_bird(90)
print_wire2()
```

You should see that some of these succeeded in adding a bird (they printed `True`) and that some of them did not (they printed `False`). Check that we have not broken any of the rules that we set out for placing birds.

Now let's take a look at `add_bird`. First, notice that it's got something (`where`) in the brackets after its name. We've seen this before. It's a special sort of *variable* called a *parameter* and in this case it's standing-in for a *value* that we will have to supply when we want to use add_bird. If we just type add_bird() we will get an error. Better still, if you are in the python shell and you type `add_bird(` a little box will appear with the words `(where)` in it, to give you a great big helpful clue about what the python shell is expecting you to type next. Just to be clear: it is not expecting you to type in `where`; it is expecting you to tell it what *value* you want to use for `where`.

If you didn't notice the little box appear when you were trying out `add_bird`, Try it now.

If you are unclear about any of this, ask for help.

Next, we have 3 `if` statements, one inside the other. We need all of them to be true in order to reach that piece of code where the comment says `# yes!` These three `if`s are checking to see whether a bird will fit.

Let's look at those `if`s, one at a time.

We are trying to place a bird at the position represented by the value of `where` – it could be anywhere on the wire. Wherever it is, we only allow the body of the bird to go there if there is no other bird or bird wing there. In other words, we need a clear piece of wire to land on. That is exactly what the first `if` is doing: it is checking a particular place on the wire to see if it is "`-`" – empty. The code inside the `if` is *only* executed if the comparison is true (the wire is empty). In that case, we go on and do the second `if`. If the comparison is false, there's nothing more we need to check; there's no way for the bird to fit here and we give up (imagine the bird flying off, or circling around for another attempt).

The second `if` is looking at location `where-1` on the wire; think of that as being to the left of the bird's body. We need to know whether a wing can fit there. There are two different ways that the wing can fit: it can fit if the wire is empty OR it can fit if there's a wing already there.

The third `if` is looking at location `where+1` on the wire; think of that as being to the right of the bird's body. As before, we need to know whether a wing can fit there and there are two different ways that it can fit: it can fit if the wire is empty OR it can fit if there's a wing already there.

If all three `if`s are true a bird can fit. The program changes the values of three elements on the list, to represent the body and the wings (one or other wing might already have been a wing, but that doesn't matter).

Can we fit a bird on the wire? either we can or we can't; there's no "almost". This type of yes/no result from a test or a comparison is called a Boolean value. In Python, the two possible Boolean values are named True and False (you always need the capital letter for the T and the F).

The last thing that `add_bird` does is to use a `return` which allows it to send back a value: either True or False, to show whether the bird fitted.

If you are unclear about any of this, ask for help.

Next we are going to use a random number to pick a place on the wire, and we are going to try to put a bird there. Actually, we are going to try to put 200 birds on the wire (we know they won't all fit!) Of course, for this we will need a loop.

Modify your program to look like the one below. You don't need to type it all in; just the parts that have changed. The new code is shown in red.

When you type the code in, your code will not be coloured like this!

```
# Birds on a wire
#

from random import randint

# wire[0] to wire [131]
wire_length = 132
wire = ["x"]*wire_length

def clear_wire():
    for i in range(0,wire_length):
        wire[i] = "-"

def print_wire():
    for i in range(0,wire_length-1):
        print(wire[i], end="")

def print_wire2():
    print("".join(wire))

# Try to add a bird at the position 'where'. If it's possible, return True
# and add the bird. If it's not possible, return False
def add_bird(where):
    if wire[where] == "-":
        if wire[where-1] == "-" or wire[where-1] == "^":
            if wire[where+1] == "-" or wire[where-1] == "^":
                # yes!
                wire[where] = "O"
                wire[where-1] = "^"
                wire[where+1] = "^"
                return True
    return False

# How many birds will fit on the wire?
def fill_wire():
    clear_wire()
    fitted = 0
    # try 200 times
    for attempt in range(0,200):
        # for wire_length=4
        # 0123
        # can't put a body at 0 or 3
        where = randint(1,wire_length-2)
        if add_bird(where):
            fitted = fitted + 1
    return fitted
```

As before, click `run->run module` (or press F5). When the program runs it should do just what it did before: nothing! As before ,there should be no errors. If you get an error, compare the code above carefully with what you typed. Correct any mistakes and try again.

After you have run the program, click in the Python Shell window and type in some commands by hand to test it out:

```
fill_wire()
print_wire2()
fill_wire()
print_wire2()
fill_wire()
print_wire2()
```

As you run it over and over again you should find different number of birds fitting on the wire. Look at the wire. Is it full (are there any places that a bird could fit). If there are, the program hasn't been trying hard enough to fill the wire. Change "`200`" to a larger number (say, `400` or `700`) and run it again.

Look at the procedure fill_wire and see if you can follow how it works. We start with some set-up, and then there's a loop. Each time around the loop, we generate a random number and put it into a variable named `where`. Next, we say `add_bird(where)` which attempts to put a bird at the place on the wire represented by the value of `where`. Remember that `add_bird()` returns `True` or `False` and we have an if that uses that result to increment (add 1 to) a variable named fitted. So, we count the number of birds that we successfully fitted on the wire. The procedure ends by sending back the value of `fitted`: the total number of birds that were fitted onto the wire.

If you are unclear about any of this, ask for help. Otherwise, read on for some extras...

Extras

1. Write a loop to call fill_wire() 100 times. After each time, print the number of birds that fitted on the wire, and print the wire.

2. Add a new variable called max_birds and keep track of the largest number of birds that you can get to fit; print the number at the end of your loop.

Hint: your code will include a line something like this:

```
if (fitted_this_time > max_birds):
      max_birds = fitted_this_time
```

3. (You don't need a computer for this one) what is the theoretical maximum number of birds that will fit on a wire of length *n* – if they were packed in neatly? Did your model ever achieve this?

Hint: work it out for wires of length 3, 4, 5, 6 and infer a general rule.

4. If your loop never achieved the theoretical maximum, change the loop so that it runs *until it finally gets to the theoretical maximum*, and count (and print out) how many attempts it takes to get there.

Hint: you might want to speed things up by not printing the wire each time

5. In `add_bird` we didn't really need three `if` statements. I the same way that we glued two comparisons together (a wing check) using `or`, we could have glued the tests for the left wing, body and right wing together using an `and`. These `and` and `or` operators are called Boolean expressions; named after a British mathematician called George Boole.

6. Birds on a wire is a 1D problem. If it seems too easy, think about how to solve the 2D problem: helicopters in a field.

Hint: you need to use a 2-dimensional list: `field=[ [" "]*10 for i in range(10)]` will create a 10x10 grid which you can access as `field[0][0]`, `field[1][2]` and so on.