Repeating operations in loops

For loops

Basics

The basic principle with a `for` loop is to run a series of commands a fixed number of times, by incrementing a counter or iterating over an array or list. A for loop can therefore be used to produce a particular action a set number of times.

Example

Here the `for` loop defines a variable `c` and uses this as a counter across a range from 0 to 10.

```
for c in range(0,10):
    print(c)
```

Did you notice that 10 isn't included? When we specify `range(a,b)` it goes from `a` up to but not including `b`. Can you edit the code so 10 is included?

Example

You can also step through a range at a different rate by passing a third parameter to the range() function. For example, this code counts in threes to 100.

```
for c in range(0,100,3):
    print(c)
```

Example

You can also count backwards through a range. Notice in the following example that the start and end parameters are the correct way round for counting down.

```
for c in range(10,0,-1):
    print(c)
```

Example

Instead of using a loop counter, you can also ask a for loop to loop over the entries in a list. Here is an example where a variable `name` is used to iterate over a list of strings.

```
characters = ["The Cat", "The Fish", "Thing 1", "Thing 2", "Sally", "Sally's b
for name in characters:
    print(name)
```

If it is important to know where you are in the list, you can use a loop counter to loop over a list, for example:

```
characters = ["The Cat", "The Fish", "Thing 1", "Thing 2", "Sally", "Sally's b
for i in range(0,len(characters)):
    print(f"character {i} is {characters[i]}")
```

Example

Of course, it is often useful to include more sophisticated code inside your loop than just outputting everything. For example, the code below iterates over a list and outputs only those numbers which are even using the remainder operator `%` (this examines `a % b` and returns the remainder from dividing `a` by `b`; for example, if the remainder is zero then we know that `b` divides `a`).

```
for i in [1,6,17,23,32,45,46,49,120,1234567890]:
    if i % 2 == 0:
        print(i)
```

Nested for loops

A particularly useful operation with for loops is to have one nested inside another. The following code illustrates this. The outer loop runs through the counter `i`, but for every iteration of `i`, the inner loop iterates through all values of `j`. Run and and see the result.

```
for i in range(0,10):
    for j in range(0,10):
        print(f"This is i={i} and j={j}")
```

Example

See if you can work out what this code is doing, then run it and see if you are right.

```
characters = ["The Cat", "The Fish", "Thing 1", "Thing 2", "Sally", "Sally's b
for name1 in characters:
    for name2 in characters:
        if name1 != name2:
            print(f"{name1} shakes hands with {name2}")
```

Example

See if you can work out what this code is doing, then run it and see if you are right.

```
for i in range(1,11):
    thisline = ""
    for j in range(1,11):
        thisline = f"{thisline}{i*j}\t"
    print(thisline)
```

Example

Here we use nested for loops to loop over a two-dimensional list. The first for loop goes through the main list, then each sub-list is looped through in the inner for loop. See if you can work out what this code is doing, then run it and see if you are right.

```
lines = [[4,9,2],[3,5,7],[8,1,6]]
for i in range(0,len(lines)):
    output = f"Row {i+1}:"
    for item in lines[i]:
        output = f"{output} {item}"
    print(output)
```

While loops

A `while` loop is a different kind of loop. Previously we saw `for` loops, which run a series of commands a set number of times. A `while` loop runs a series of commands until a certain condition is met.

An example of a simple `while` loop is given below. This uses a variable `i` and keeps going until $i^2 \geq 100$.

```
i = 1
while i**2 < 100:
  print(f"{i} squared is {i**2}")
  i += 1
```

Notice that we have to manually increment a counter using `i += 1` because the loop doesn't do this for us (unlike a `for` loop).

Notice that we did not need to know that the loop was going to stop at `i=9` in advance - the `while` condition worked this out for us as we went.

As well as using a counter, you can use run a while loop over a list, which tells it to run while there are still items in the list. Inside your list, you need to be removing items from the list (or the list will run forever).

For example, this while loop keeps going over a list, checking the first character of the first item and either deleting it from the list or popping it onto a different list, until the variable `non_t_franchises` contains all the entries that don't start with 'T'.

```
franchises = ["Star Wars", "Alien", "Harry Potter", "Star Trek", "Marvel Cinem
"Batman", "X Men", "James Bond", "The Lord of the Rings", "Jurassic Park", "DC
"The Fast and the Furious", "Transformers", "Pirates of the Caribbean", "The H
"Twilight", "Mission: Impossible", "Indiana Jones"]
```

```
non_t_franchises = []

while franchises:
    if franchises[0][0].lower() == "t":
        del franchises[0]
    else:
        non_t_franchises.append(franchises.pop(0))

print(f"The franchises that don't start with 'T' are: {non_t_franchises}")
```

Break

In general, a `while` loop stops when the `while` condition is first not met. However, if you want to stop the loop early you use the command `break`.

In the example below, a ball is randomly chosen to be removed from a list (potted) until there are no balls left, then the order of balls potted is displayed. However, if the black is potted then the loop stops early using `break`. Run it a few times to see it in action.

```
import random
balls = ["yellow","blue","red","purple","orange","green","brown","black"]
potted = []
random.shuffle(balls)
while balls:
    thisball = balls.pop()
    potted.append(thisball)
    if thisball == "black":
        break
print(f"Balls potted: {potted}")
```

In this way, you can just run a `while` loop forever until some condition is met, perhaps if you want the test to come in the middle of the loop rather than at the start of each iteration.

```
i=0
while True:
    i=i+1
    if i**2 > 99:
        break
    print(f"{i} squared is {i**2}")
```

Continue

The command `continue` is used to tell Python to stop running the commands inside the `while` loop and skip to the next iteration of the loop.

Here is a simple example that prints out the numbers from 1 to 20 that aren't divisible by 5. The `continue` is used to skip the rest of this iteration of the loop (i.e. it doesn't do `print(i)`) and start the next time through the loop.

```
i = 0
while i < 21:
    i = i + 1
    if i % 5 == 0:
        continue
    print(i)
```