Simple Loops

Iteration / Looping

We will often want to write pieces of code that run multiple times.

This is called *iteration* or *looping*. A piece of code that repeats is called a *loop*.

While Loops

```
while boolean-expression:
    ... do something ...
```

The code block is run repeatedly, so long as the expression is True.

The expression is tested at the start of each iteration. Once the expression becomes False, the loop terminates.

Something in the loop needs to make the expression become False eventually! An *infinite loop* occurs when the expression is always True.

While Loops

```
password = input("Enter the password: ")
while password != "swordfish":
    password = input("Wrong password! Try again: ")
print("Correct password entered.")
```

If the password is entered correctly the first time, then the loop is never run!

Exercise: Enter a number between 1 and 10

```
password = input("Enter the password: ")
while password != "swordfish":
    password = input("Wrong password! Try again: ")
print("Correct password entered.")
```

Adapt the example to ask the user for a number between 1 and 10. The program should keep asking until it gets a value within the range.

Exercise: Enter a number between 1 and 10

```
    number = int(input("Enter a number between 1 and 10: "))
    while number < 1 or number > 10:
        number = int(input("Number not in range! Enter a
number between 1 and 10: "))
    print("You entered: %d." % number)
```

(You could also input a float rather than an int.)

Counting

```
counter = 1
while counter <= 10:
    print(counter)
    counter = counter + 1
```

Modifying a Variable - Alternative Syntax

- `+=` Adds a value to the variable.
- `-=` Subtracts a value from the variable.
- `*=` Multiplies the variable by a value.
- `/=` Divides the variable by a value.

Modifying a Variable - Alternative Syntax

- `+=` Adds a value to the variable.
- `-=` Subtracts a value from the variable.
- `*=` Multiplies the variable by a value.
- `/=` Divides the variable by a value.

```
counter = 1
while counter <= 10:
    print(counter)
    counter += 1
```

Exercise: Multiples

Write a program that asks the user for a number, and outputs a list of multiples of that number. e.g. For input 8, the multiples would be 8, 16, 24, 32, etc...

You could just output the list of numbers, but bonus points for formatting the output nicely.

Exercise: Multiples

Write a program that asks the user for a number, and outputs a list of multiples of that number. e.g. For input 8, the multiples would be 8, 16, 24, 32, etc...

You could just output the list of numbers, but bonus points for formatting the output nicely.

```
number = int(input("Enter a number: "))
i = 1
while i <= 12:
    print("%d * %d = %d" % (i, number, i * number))
    i += 1
```

Changing the Flow

It is possible to change the flow of a loop without making the loop condition False.

Doing this is considered inelegant, though!

Nonetheless, this can sometimes be useful.

Breaking out of the loop

To terminate a `while` loop early, use the `break` command.

```
i = 1
while True:
    print(i)
    if i == 5:
        break
    i += 1
```

Skipping an iteration of the loop

To stop the current iteration and go straight to the next iteration, use `continue`.

```
i = 0
while i < 10:
    i += 1
    if i == 7:
        # Skip printing number 7
        continue
    print(i)
```

Where are we now?

We now have enough knowledge of Python to make much more complex programs than we could last week!

Practice makes perfect...

Exercise: Factoring Numbers

Write a program that asks the user for an integer, then list all the factors of that integer.

- A factor of an integer is a number that divides wholly into it. For example, the factors of 12 are 1, 2, 3, 4, 6, and 12.
- The modulo operator % gives the remainder when one number is divided by another. Work out how to use this to test for factors.

Exercise: Factoring Numbers

```
number = int(input("Enter a number: "))
i = 1
while i <= number:
    if number % i == 0:
        print(i)
    i += 1
```

How might we make this more efficient?

Exercise: Factoring Numbers

```
number = int(input("Enter a number: "))
i = 1
while i <= number ** 0.5:
    if number % i == 0:
        print("%d, %d" % (i, number / i))
    i += 1
```

Exercise: Multiplication Table

Write a program that produces a multiplication table: a two-dimensional grid of multiplications.

- You will need two loops, one nested inside the other.
- To output without automatically adding a newline to the end, use `print(expression, end="")`.
- To only print a newline, use `print()`.
- To line up values in printed output, you can use a tab character. To include a tab character in a string, use "`\t`". The `\t` will be replaced with a tab when the program is run.

Exercise: Multiplication Table

```
i = 1
while i <= 12:
    j = 1
    while j <= 12:
        print("\t%d" % (i * j), end="")
        j += 1
    print()
    i += 1
```

Exercise: Number Guessing Game

Write a program that generates a random integer, then gives you a limited number of attempts to guess it. Each time you guess wrongly, the program should tell you whether you were too high or too low.

- To be able to generate a random integer, you will need to type the following at the top of your program: `from random import randint`.
- What exactly this does will be the subject of a future lesson - for now, just add it in!
- The function `randint(1, 100)` can then be used to generate a random integer between 1 and 100 (for example).
- You will want to save the random number to a variable! Repeated calls to `randint` will always give different numbers.

Exercise: Number Guessing Game

```
from random import randint
number = randint(1, 100)
attempts = 0
done = False
while not done and attempts < 10:
    guess = int(input("Guess a number: "))
    if guess == number:
        print("You win!")
        done = True
    elif guess < number:
        print("Too low!")
    else:
        print("Too high!")
    attempts += 1
if not done:
    print("You lose! My number was %d." % number)
```

Summary

- Flow control
  - Decisions
  - Loops

How can we manipulate more complicated data, rather than just individual variables?