Simple Loops

Iteration

- Often, we will want a piece of code to run multiple times.
- This is called *iteration* or *looping*.
- A specific piece of code that is run multiple times is called a *loop*.

While Loop

- The most general form of loop is a *while loop*.
- A while loop looks a lot like an if statement: it has a boolean expression and a block of code. Its syntax is:
    - ```
      while boolean-expression:
          ... do something ...
      ```
- The while loop tests the boolean expression, and if it is true, it executes the code block. Once the code block is finished, the while loop tests the expression again, and if it is true, executes the code block again. And so on…
- The code block stops being looped once the expression becomes false.
    - Something in the code block needs to make the expression eventually become false, or the loop will go on forever!
    - This is an *infinite loop*, and will cause your program to hang.
    - If you do manage to enter an infinite loop with any of your programs, press Ctrl+C to terminate the program.
- Example:
    - ```
      password = input("Enter the password: ")
      while password != "swordfish":
          password = input("Wrong password! Try again: ")
      print("Correct password entered.")
      ```
    - The while loop will run the code block so long as the correct password has not been entered. Once the correct password is entered, the loop finishes and the program can continue.
    - Note that if the correct password is entered immediately, then the boolean expression is immediately false - so the loop never gets run.
- Exercise:
    - Adapt the above example to ask the user for a number between 1 and 10.
    - If a number outside this range is entered, the program should ask again, and keep asking until a valid number is given.
    - Once a number between 1 and 10 has been entered, the program can stop.
    - Answer:
        - ```
          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)
          ```

Counting with While Loops
          ```

- Often, we might want to use a counter variable with our while loop as part of the terminating condition.
- We will *initialise* the counter with a starting value.
- Each iteration of the loop, we add to (or subtract from) the counter.
- The loop repeats while the counter is within a certain range.
- Once the counter leaves that range, the loop ends.
- Example:
    - ```
      counter = 1
      while counter <= 10:
          print(counter)
          counter = counter + 1
      ```
    - This will output the numbers 1 to 10.
    - The counter is incremented by calculating the value 1 higher than the counter, then reassigning the counter with that value.
    - This is cumbersome, though.
- There is alternative syntax for modifying a variable with itself:
    - `+=` Adds a value to the variable.
    - `-=` Subtracts a value from the variable.
    - `*=` Multiplies the variable by a value.
    - `/=` Divides the variable by a value.
- So our example could have been:
    - ```
      counter = 1
      while counter <= 10:
          print(counter)
          counter += 1
      ```
    - This is neater!
- Exercise:
    - Output times tables for a given value.
    - i.e. Take user input of a number and output multiples of that number.
    - You could just output the list of numbers, but labelling the calculations (e.g. “2 * 8 = 16” rather than just “16”) might be nicer.
    - Sample answer:
        - ```
          number = int(input("Enter a number: "))
          i = 1
          while i <= 12:
              print("%d * %d = %d" % (i, number, i * number))
              i += 1
          ```

<u>Breaking Out</u>

- It is possible to terminate a while loop without making the condition false.
- It is also possible to return to the top of the loop without reaching the end of the code block.
- Both such things are considered inelegant by some - it is easier to reason about your loops, and be sure they work correctly, if you base their looping solely on the condition.
          ```

- Nonetheless…
- To exit a while loop early, use the `break` command.
- Example:
  - ```
    i = 1
    while True:
        print(i)
        if i == 5:
            break
        i += 1
    ```
  - This would have been better achieved using a sensible looping condition.
- To return immediately to the top of the loop's code block, use the `continue` command.
- Example:
  - ```
    i = 0
    while i < 10:
        i += 1
        if i == 7:
            # Skip printing number 7
            continue
        print(i)
    ```
  - This would have been better achieved by using the if statement to decide whether to call `print` or not.
- These may have their occasional uses, and it is important to know they exist - but in general, you will have little use for them.

Pre-Exercises Summary

- We have now covered basic control flow in Python: making decisions and using loops.
- Along with what we previously learnt about expressions and variables, this is enough knowledge to write some much more useful, interesting, and complicated programs than what we could do at the end of last week.
- The rest of the session will be a series of exercises to allow you to put these skills to use.

Exercise: Factoring Numbers

- Write a program that asks the user for an integer, then list all the factors of that integer.
- Hints:
  - 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.
- Sample answer:
  - ```
    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?
    - We can get factors in pairs, by dividing the number by one of the factors to get its complement.
    - So we only need to loop up to the square root of the number.
    - ```
      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.
- Hints:
    - 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.
- Sample answer:
    - ```
      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 will tell you whether you were too high or too low.
- Hints:
    - 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.
- Sample answer:
    - ```
      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)
      ```
- What is the optimal strategy for playing this game?
    - Binary search.

Summary

- We can now control the flow of execution of our programs.
- This lets us write programs that can make useful decisions!
- With the basic of Python under our belts, we can move on to more advanced features that will let us write code that is more powerful and capable.

Next lesson: Compound Data Types
      ```