# Table of Contents

## Repetition with `while`

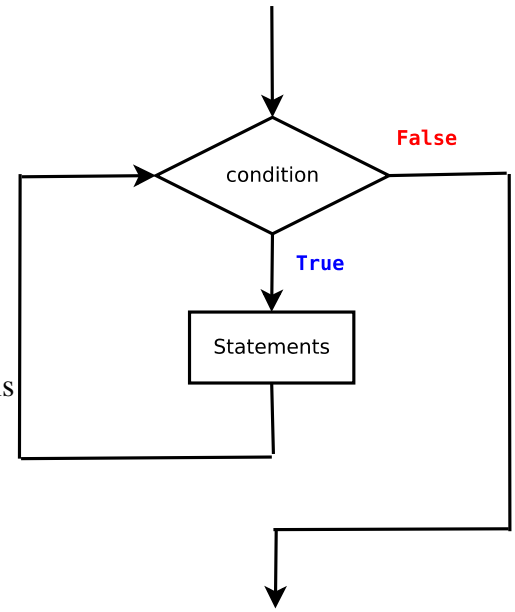
A `while` loop allows a block of statements to be executed repeatedly, while a specified condition is `True`.

The syntax of a while loop is:

```
while condition:
    statements
```

must end with `:`

indented




The *`condition`* is a Boolean expression.

*`statements`* is a sequence of one or more statements; this repeatedly executes as long as the condition remains **`True`**.

When the condition is **`False`**, the loop terminates.

### `while` Loop Applications

while loops are used in the following situations:

1.As a **counting** loop: count how many times the statements have been executed.
2.**User-controlled** loop: repeat until the user decides to stop;
3.**Sentinel-controlled** loop: repeatedly input data of until a special value is provided;
4.**Input validation** loop: input a value until a valid value within the valid range is entered;
5.**General conditional loop**: repeated processing until a desired condition is met.

## Counting `while` Loops

A counting loop is used to repeat a block of code a set number of times. The program uses a variable to count how many times the code has been repeated, and stops when the required number of repetitions has been achieved.

*Example*

Every day, a weather station receives 6 temperatures expressed in degrees Fahrenheit. A program inputs each Fahrenheit temperature, converts it to Celsius and displays the Celsius temperature on screen.

*Specification Table*

| Inputs | Processing | Outputs |
|---|---|---|
| fahrenheit temperature | Input fahrenheit temperature<br>Convert to celsius<br>Display celsius temperature | celsius temperature |

Formula to convert Fahrenheit to Celsius: $\frac{5}{9}\times(Fahrenheit-32)$

Here's the program:

```python
print("This program converts Fahrenheit temperatures to Celsius")

#Input fahrenheit temperature
fahrenheit = float(input("Enter the Fahrenheit temperature: "))

# Convert to Celsius
celsius = 5/9 * (fahrenheit - 32)

# Print Celsius temperature
print(f"The temperature in Celsius is: {celsius:.1f}")
```

*Sample Output*

```
Enter the Fahrenheit temperature: 78
The temperature in Celsius is: 25.6
```

Each time it runs, the program inputs a single Fahrenheit temperature, converts it to Celsius and displays the result. You could run the program 6 times, to convert the 6 values. But instead, a while loop can be used: repeating the code to input, convert and display a temperature 6 times.

***Counting Loop Logic***

The purpose of a counting loop is to repeat a block of code a set number of times. It does this by counting how many times the code has been repeated, and stopping when the required number of repetitions has been done.

The logic is as follows:

- Start the count at 0
- Check if the count has reached the target number if it has, then stop
- If the count hasn't reached the target number, then repeat the block of statements and increase the count by 1

This is represented in ***pseudo-code*** as follows:

```
count = 0

while count < target
    execute statements

    count = count + 1
```

Pseudo-code is not Python. It describes the logic of the solution, but is not the actual code.

The line

```
count = count + 1
```

indicates that the count should be increased by 1. The assignment statement using = means "take what's on the right-hand side and store it in the variable on the left"

In this case, the right-hand side is the current value of count with 1 added to it. This is then stored back in the `count` variable.

Suppose the number of repetitions needed is 6. At the start, `count` is 0. Each time it is increased by 1.

| **count** | **Check: count < 6** | **Process temperature** | **Add 1 to count** |
|---|---|---|---|
| 0 | FÍOR | 56 | 1 |
| 1 | FÍOR | 73 | 2 |
| 2 | FÍOR | 64 | 3 |
| 3 | FÍOR | 80 | 4 |
| 4 | FÍOR | 67 | 5 |
| 5 | FÍOR | 77 | 6 |
| 6 | FALSA | | |

The variable `count` keeps track of how many repetitions have been done. It starts at 0. The while loop checks if the value of `count` is less than 6, the number of repetitions required. If it is, then the actions in the while loop are performed. Before the end of the while loop, the `count` variable is increased by 1, and the while loop again checks if the value of count is less than 6.

Finally, when the value of `count` reaches 6, the `while` loop stops, and control moves to the next statement after the `while` loop (if any).

**Counting Loop: Python Syntax**

In a ***counting loop*** repetition of the loop is managed by a variable whose value represents a *count*.

A counting loop follows this general format:

```
count = 0
while count < num_repetitions:
    # do some processing
    count = count + 1
```

The `count` represents the number of repetitions so far. It starts at 0, so the while loop condition is **`True`**. Each time the loop executes, the `count` is increased.

When it reaches `num_repetitions` the while loop condition is **`False`** and the while loop stops.

**Version 1: Counting Loop**

Here's the Python program for the temperature conversions, using a counting loop.

```python
print("This program converts Fahrenheit temperatures to Celsius")

# start the count at zero
count = 0

# keep going until the count reaches 6
while count < 6:
    #Input fahrenheit temperature
    fahrenheit = float(input("Enter the Fahrenheit temperature: "))

    # Convert to Celsius
    celsius = 5/9 * (fahrenheit - 32)

    # Print Celsius temperature
    print(f"The temperature in Celsius is: {celsius:.1f}")

    # inbcrease the count
    count = count + 1

print()
print("All temperatures processed")
```

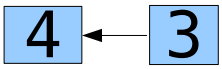
*Sample Run*

```
This program converts Fahrenheit temperatures to Celsius

Enter the Fahrenheit temperature: 90
The temperature in Celsius is: 32.2

Enter the Fahrenheit temperature: 45
The temperature in Celsius is: 7.2

Enter the Fahrenheit temperature: 104
The temperature in Celsius is: 40.0

Enter the Fahrenheit temperature: 56
The temperature in Celsius is: 13.3

Enter the Fahrenheit temperature: 23
The temperature in Celsius is: -5.0

Enter the Fahrenheit temperature: 12
The temperature in Celsius is: -11.1

All temperatures processed
```

**Shorthand +=**

The count variable is increased by 1 each time the loop body is repeated:

```
count = count + 1
```




On the right-hand-side, the current value of count has 1 added to it, and then the result is assigned back to the `count` variable.

In Python, there's short-hand for this is

```
count += 1
```

which means:

add 1 to the value of `count`
store the result back in the variable `count`

There's no ++ operator in Python

Similarly,

```
total += num_hits
```

means:

add the value of `num_hits` to the value of `total`
store the result back in the variable `total`

*Watch Out*:

A common mistake is to forget to increase the count variable:

```
count = 0
while count < 10:
    print(count)
```

This will result in an infinite loop, i.e. it will never stop (by itself).

```
0
0
0
0
0
0
0
0
0
0
```

## User-Controlled Loops

A **`while`** loop can be used to repeat a block of code until the user chooses to stop.

The general approach is:

- select a value which signifies the user will continue, e.g. "y"
- set a variable to this value e.g. `choice = "y"`
- while loop condition tests if `choice` equals this value

  `while choice == "y":`
- within the loop, ask the user if s/he wants to continue
- when the user's choice is not `"y"` the loop stops

**Version 2.1: User-Controlled Loop**

This version of the Temperatures program demonstrates a user-controlled loop:

```
print("This program converts Fahrenheit temperatures to Celsius")

# assume the user wants to convert at least 1 temperapture
choice = "y" # for "yes"

# keep going while the user's choice is "y"
while choice == "y":
    #Input fahrenheit temperature
    fahrenheit = float(input("Enter the Fahrenheit temperature: "))

    # Convert to Celsius
    celsius = 5/9 * (fahrenheit - 32)

    # Print Celsius temperature
    print(f"The temperature in Celsius is: {celsius:.1f}")

    # ask the user if s/he wants to convert another temperature
    choice = input("Convert another temperature (y/n)? ")

print()
print("All temperatures processed")
```

*Sample Output*

```
This program converts Fahrenheit temperatures to Celsius

Enter the Fahrenheit temperature: 90
The temperature in Celsius is: 32.2

Convert another temperature (y/n)? y

Enter the Fahrenheit temperature: 45
The temperature in Celsius is: 7.2

Convert another temperature (y/n)? y

Enter the Fahrenheit temperature: 104
The temperature in Celsius is: 40.0

Convert another temperature (y/n)? y

Enter the Fahrenheit temperature: 56
The temperature in Celsius is: 13.3

Convert another temperature (y/n)? y

Enter the Fahrenheit temperature: 23
The temperature in Celsius is: -5.0

Convert another temperature (y/n)? y

Enter the Fahrenheit temperature: 12
The temperature in Celsius is: -11.1

Convert another temperature (y/n)? n

All temperatures processed
```

*How it Works*

The loop works by initialising the `choice` variable to `"y"`. Before executing the block of statements associated with it, the while loop checks if `choice` is equal to `"y"`. If it is, then the block of statements are executed: the Fahrenheit temperature is input, converted to Celsius and the output displayed. The user is then asked if the want to convert another value, and their input is stored in `choice`. Whenever the user enters a value that is not `"y"`, the `while` loop condition will be `False`, and so the loop will terminate.

| **choice** | **Check: choice == "y"** | **Input Fahrenheit** | **Output Celsius** | **Input choice** |
|---|---|---|---|---|
| y | FÍOR | 90 | 32.2 | y |
| y | FÍOR | 45 | 7.2 | y |
| y | FÍOR | 104 | 40 | y |
| y | FÍOR | 56 | 13.3 | y |
| y | FÍOR | 23 | -0.5 | y |
| y | FÍOR | 12 | -11.1 | y |
| n | FALSA | | | |

**Version 2.2: User-Controlled Loop (boolean variable)**

The following version uses a boolean variable `finished` to check whether or not to repeat the while loop block.

```python
print("This program converts Fahrenheit temperatures to Celsius")

# assume the user wants to convert at least 1 temperapture
finished = False # for not finished

# keep going until the user is finished
while not finished:
    #Input fahrenheit temperature
    fahrenheit = float(input("Enter the Fahrenheit temperature: "))

    # Convert to Celsius
    celsius = 5/9 * (fahrenheit - 32)

    # Print Celsius temperature
    print(f"The temperature in Celsius is: {celsius:.1f}")

    # ask the user if s/he is finished
    response = input("Are you finished (y/n)? ")
    if response.lower() == "y":
        finished = True

print()
print("All temperatures processed")
```

*Sample Output*

```
This program converts Fahrenheit temperatures to Celsius

Enter the Fahrenheit temperature: 90
The temperature in Celsius is: 32.2

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 45
The temperature in Celsius is: 7.2

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 104
The temperature in Celsius is: 40.0

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 56
The temperature in Celsius is: 13.3

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 23
The temperature in Celsius is: -5.0

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 12
The temperature in Celsius is: -11.1

Are you finished (y/n)? y

All temperatures processed
```

*How it Works*

The variable `finished` is initialised to `False`, meaning the user is not finished. The `while` loop then checks if the user is not finished, using the condition `not finished`. If `not finished` is `True`, then the loop block is executed, otherwise the loop terminates.

Within the loop, the user is asked if s/he is finished. If s/he is, then `finished` is set to `True`.

| finished | Check: not finished | Input Fahrenheit | Output Celsius | Input response | response == "y" |
|---|---|---|---|---|---|
| FALSE | FÍOR | 90 | 32.2 | n | FALSE |
| FALSE | FÍOR | 45 | 7.2 | n | FALSE |
| FALSE | FÍOR | 104 | 40 | n | FALSE |
| FALSE | FÍOR | 56 | 13.3 | n | FALSE |
| FALSE | FÍOR | 23 | -0.5 | n | FALSE |
| FALSE | FÍOR | 12 | -11.1 | n | FALSE |
| TRUE | FALSA | | | | |

**`while` loop: General Principle**

In each of the `while` loops, there are three parts:

1.***Initialisation***:

Before the loop, set a *`variable`* to a starting value

`count = 0`  `choice = "y"`  `finished = False`

2.***Testing***:

At the start of the loop, check the value of the *`variable`*

`count < 6`  `choice == "y"`  `not finished`

3.***Updating***:

Before the end of the loop, allow the value of the *`variable`* to be changed

`count += 1`  *input* `choice`  `finished = True`

### Infinite Loop and break

A simpler form of while loop is

```
while True:
```

The condition is always True, so the loop will repeat indefinitely. To exit the loop, use the break statement: control moves to the next statement after the loop.

**Version 2.3: User-Controlled Loop (infinite loop / break)**

This version of the program demonstrates an infinite while loop.

```
print("This program converts Fahrenheit temperatures to Celsius")

# keep going until the user is finished
while True:
    #Input fahrenheit temperature
    fahrenheit = float(input("Enter the Fahrenheit temperature: "))

    # Convert to Celsius
    celsius = 5/9 * (fahrenheit - 32)

    # Print Celsius temperature
    print(f"The temperature in Celsius is: {celsius:.1f}")

    # ask the user if s/he is finished
    response = input("Are you finished (y/n)? ")
    if response.lower() == "y":
        break

print()
print("All temperatures processed")
```

*Sample Output*

```
This program converts Fahrenheit temperatures to Celsius

Enter the Fahrenheit temperature: 90
The temperature in Celsius is: 32.2

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 45
The temperature in Celsius is: 7.2

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 104
The temperature in Celsius is: 40.0

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 56
The temperature in Celsius is: 13.3

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 23
The temperature in Celsius is: -5.0

Are you finished (y/n)? n

Enter the Fahrenheit temperature: 12
The temperature in Celsius is: -11.1

Are you finished (y/n)? y

All temperatures processed
```

*How it Works*

The loop will repeat indefinitely, because the loop condition is always True.

The loop terminates using the break keyword, which stops the loop; program control moves to the next statement after the loop. This is triggered when the user indicates that s/he is finished.

## Sentinel-Controlled Loop

A **Sentinel-controlled** loop is used to repeatedly input data until a special value is provided, called the "sentinel". It avoids having to ask the user if s/he is finished, or wants to add more data. By inputting the special "sentinel" value, s/he indicates s/he's finished. The program then terminates the loop.

In most cases, a sentinel value of -1 is useful as it cannot be mistaken for a valid data value. However, with temperature data, -1 could be a valid value, so the sentinel for this program will be 9999.

**Version 3: Sentinel-Controlled Loop**

This version of the Temperatures program demonstrates a Sentinel-controlled while loop.

```
print("This program converts Fahrenheit temperatures to Celsius")

# keep going until the user is finished
while True:
    #Input fahrenheit temperature
    fahrenheit = float(input("Enter the Fahrenheit temperature or 9999 to finish: "))

    # check if s/he is finished
    if fahrenheit == 9999:
        break

    # Convert to Celsius
    celsius = 5/9 * (fahrenheit - 32)

    # Print Celsius temperature
    print(f"The temperature in Celsius is: {celsius:.1f}")


print()
print("All temperatures processed")
```

*Sample Output*

```
This program converts Fahrenheit temperatures to Celsius

Enter the Fahrenheit temperature or 9999 to finish: 90
The temperature in Celsius is: 32.2

Enter the Fahrenheit temperature or 9999 to finish: 45
The temperature in Celsius is: 7.2

Enter the Fahrenheit temperature or 9999 to finish: 104
The temperature in Celsius is: 40.0

Enter the Fahrenheit temperature or 9999 to finish: 56
The temperature in Celsius is: 13.3

Enter the Fahrenheit temperature or 9999 to finish: 23
The temperature in Celsius is: -5.0

Enter the Fahrenheit temperature or 9999 to finish: 12
The temperature in Celsius is: -11.1

Enter the Fahrenheit temperature or 9999 to finish: 9999

All temperatures processed
```

*How it Works*

This program also uses an infinite loop. It repeatedly displays a prompt, and waits for the user's input. If the user inputs the sentinel value, 9999, the `break` statement is used to terminate the loop. Otherwise the temperature value is converted and the loop is repeated.

**Application: Repeating Menu**

An example of a sentinel-controlled `while` loop is where a program repeatedly displays a menu. The program continues until the user enters a specific value indicating s/he wants to quit.

The logic is as follows:

- Display the menu
- Input the user's choice
- If s/he chooses to quit, then do
- Otherwise process his/her choice

The menu could offer to provide system statistics, depending on the number chosen by the user.

```
System Statistics
----------------
1. Uptime
2. Disk Space
3. Memory
Enter your choice or 0 to quit
```

**Python Program**

```python
# Program to perform basic System Administration Tasks
from subprocess import getoutput

while True: # keep going indefinitely
    print() # Blank line before the menu
    print("1.Disk Free Information")
    print("2.Memory")
    print("3.Uptime")
    choice = int(input("Enter your choice of 0 to quit: ")  )

    if choice == 0:
        break

    elif choice == 1: # Disk Free
        print(getoutput("df -h /"))

    elif choice == 2: # Memory
        print(getoutput("free -g"))

    elif choice == 3: # Uptime
        print(getoutput("uptime"))

    else:
        print("Invalid input, try again")
```

*Sample Output*

```
System Statistics
----------------
1. Uptime
2. Disk Space
3. Memory
Enter your choice or 0 to quit 1
 17:44:46 up  1:46,  0 users,  load average: 0.78, 0.73, 0.74

System Statistics
----------------
1. Uptime
2. Disk Space
3. Memory
Enter your choice or 0 to quit 3
             total       used       free     shared    buffers     cached
Mem:          4012       1771       2241          0        296        709
Swap:         3812          0       3812

System Statistics
----------------
1. Uptime
2. Disk Space
3. Memory
Enter your choice or 0 to quit 0
```

*How it Works*

This program also uses an infinite loop. It repeatedly displays a menu with a list of chocies, and waits for the user's input. If the user decides to quit, the `break` statement is used to terminate the loop. Otherwise the user's choice is processed and the loop is repeated.

## Input Validation Loop

Any time a user inputs data in a program there's a possibility the s/he will enter an incorrect value. Best practice involves validating user input to ensure that the program can process the data correctly.

The logic of an input validation loop is as follows (pseudocode):

```
Input value
While value is not valid
     Print "Invalid value"
     Input value
```

For example, an IP address (version 4) takes the form w.x.y.z where w x y and z are ***octets*** in the range 0 – 255. For example, 192.168.34.10 and 89.47.31.179 are valid IP addresses. It would be a mistake if any of the octets entered was negative, or greater than 255. So, an octet is valid if 0 <= octet <= 255.

*Python Program*

Here's the Python program to validate an IP address octet:

```python
# Check for an invalid IP address octet
octet = int(input("Enter the octet: "))

# while the value is not valid
while not 0 <= octet <= 255:
    print("Invalid, try again")
    octet = int(input("Enter the octet: "))

# value is valid
print("Valid octet entered")
```

*Sample Output*

**Fourth value is valid**

```
Enter the octet: 999
Invalid, try again

Enter the octet: 259
Invalid, try again

Enter the octet: -123
Invalid, try again

Enter the octet: 10
Valid octet entered
```

**First value is valid**

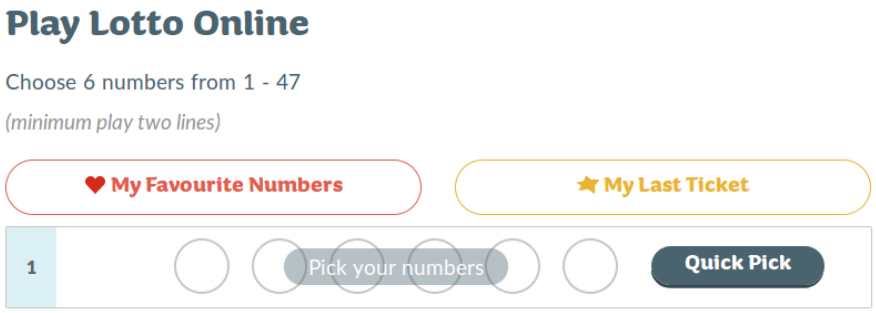
```
Enter the octet: 128
Valid octet entered
```

*How it Works*

First, the user is prompted to enter the value. The `while` loop condition checks if it is <u>not</u> valid, in which case, the user will be prompted to re-enter the value. If the value is valid, the `while` loop condition will be `False` and so the loop will terminate.

**Application: Lotto Numbers**

When playing lotto online, you pick 6 numbers between 1 and 47.




*Python Program*

This program uses a while loop to validate the users' chosen lotto numbers, i.e. to ensure that each number is between 1 and 47, inclusive.

```
print("Pick 6 Numbers from 1-47")
print("------------------------")

count = 0 # number of valid numbers picked

# keep going until 6 valid numbers have been picked
while count < 6:
    # Input the number
    print("\nNumber", count+1)
    number = int(input("Enter a number between 1 and 47: "))

    # While the number is not valid
    while not 1 <= number <= 47:
        print("Invalid number")

        # Input the number again
        number = int(input("Enter a number between 1 and 47: "))

    print("Number is valid")

    count = count + 1 # another valid number picked

print("6 Valid Numbers")
```

*Sample Output*

```
Pick 6 Numbers from 1-47
------------------------
Number 1
Enter a number between 1 and 45: 3
Number is valid

Number 2
Enter a number between 1 and 45: 5
Number is valid

Number 3
Enter a number between 1 and 45: 6
Number is valid

Number 4
Enter a number between 1 and 45: 18
Number is valid

Number 5
Enter a number between 1 and 45: 31
Number is valid

Number 6
Enter a number between 1 and 45: 45
Number is valid

6 Valid Numbers
```

```
Pick 6 Numbers from 1-47
------------------------
Number 1
Enter a number between 1 and 45: 50
Invalid number

Enter a number between 1 and 45: 0
Invalid number

Enter a number between 1 and 45: 3
Number is valid

Number 2
Enter a number between 1 and 45: 5
Number is valid

Number 3
Enter a number between 1 and 45: 6
Number is valid

Number 4
Enter a number between 1 and 45: 18
Number is valid

Number 5
Enter a number between 1 and 45: 31
Number is valid

Number 6
Enter a number between 1 and 45: 45
Number is valid

6 Valid Numbers
```

*How it Works*

The program uses two while loops:

1.A counting loop to ensure that 6 numbers have been input, and

2.An input validation loop to ensure that each number is valid.

The `count` variable starts at 0, and the counting loop will terminate when the count reaches 6. Each time the user enters a number, the input validation loop checks if it is invalid, i.e., not between 1 and 47. If it is invalid, the user is prompted to re-enter the number, and this will repeat until s/he provides a valid number. If the number is valid, the input validation loop terminates and the counting loop continues.

Notice that the `count` variable is only increased after a valid number has been entered.

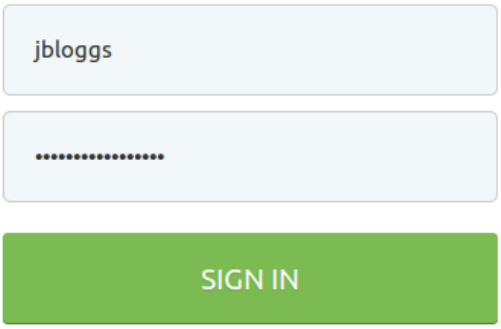
*Example of a Nested While Loop*

This is actually an example of a ***nested*** `while` loop – a while loop within another while loop.

```
print("Pick 6 Numbers from 1-47")
print("------------------------")

count = 0 # number of valid numbers picked

# keep going until 6 valid numbers have been picked
while count < 6:
    # Input the number
    print("\nNumber", count+1)
    number = int(input("Enter a number between 1 and 47: "))

    # While the number is not valid
    while not 1 <= number <= 47:
        print("Invalid number")

        # Input the number again
        number = int(input("Enter a number between 1 and 47: "))

    print("Number is valid")

    count = count + 1 # another valid number picked

print("6 Valid Numbers")
```

**Application: Simulating a User Login**

A user logging in is a form of input validation.




The program needs to check if his/her username and password are valid: that is, if they correctly match a username and password pair on the system.

**Simplistic User Login: v1 – Unlimited Attempts**

The following program is a simplistic implementation of a login program. It is not good practice to include the username and password directly in the code. The first version allows unlimited attempts.

```
# Input username and password
username = input("Username: ")
password = input("Password: ")

# keep going until the user
while username != "jbloggs" or password != "Secret123":
    print("Invalid username or password, try again")

    # Input username and password again
    username = input("Username: ")
    password = input("Password: ")

print("Login successful")
```

*Sample Output*

```
Username: jbloggs
Password: secret123
Invalid username or password, try again

Username: JBloggs
Password: secret123
Invalid username or password, try again

Username: JoeBloggs
Password: Secret123
Invalid username or password, try again

Username: jbloggs
Password: Secret123
Login successful
```

```
Username: jbloggs
Password: Secret123
Login successful
```

**Simplistic User Login: Version 2 - Max 3 Attempts**
In this version, the user is limited to 3 login attempts:

```
count = 0 # number of login attempts

while count < 3: # maximum 3 attempts
    # input username and password
    username = input("Username: ")
    password = input("Password: ")

    # check if they match the correct username and password
    if username == "jbloggs" and password == "Secret123":
        print("Login succesful")
        break # exits the loop
    else:
        print("Login failed")
        count += 1

# check: too many attempts?
if count == 3:
    print("Too many failed attempts")
    exit()

print("\nWelcome")
```

*Sample Output*

```
Username: jbloggs
Password: secret123
Login failed

Username: joebloggs
Password: secret123
Login failed

Username: joebloggs
Password: Secret123
Login failed

Too many failed attempts
```

### Python Feature: while - else

You are familiar with the `if-else` structure, used to implement decision making.

In Python, you can add an `else` to a `while` loop:

```
while condition:
    # some code
    break

else:
    # other code

# more code
```

There are two possibilities:
1.The loop completes normally (*condition* is `False`), and the code in the `else` block is executed;
2.The `break` statement exits the loop, in which case the code in the `else` block is skipped.

**Simplistic User Login: Version 3 `while - else`**

The next version of the simplistic login program demonstrates a `while-else` structure.

```python
count = 0 # number of login attempts

while count < 3: # maximum 3 attempts
    # input username and password
    username = input("Username: ")
    password = input("Password: ")

    # check if they match the correct username and password
    if username == "jbloggs" and password == "Secret123":
        print("Login succesful")
        break # exits the loop
    else:
        print("Login failed")
        count += 1
else:
    print("Too many failed attempts")
    exit()

print("\nWelcome")
```

The counting loop allows up to 3 login attempts. If the count reaches 3, the while loop condition evaluates as `False`, and the `else` block is executed, in this case, terminating the program.

If the user logs in successfully within the 3 attempts, the `break` statement within the `while` loop skips the `else` block and program control moves to the next statement after the `while` loop.

## Calculating a Total and Average using `while`

A `while` loop can be used to calculate a total and/or an average of a series of values. To calculate a total, you need to input each value and add it to the total, one value at a time. To calculate an average, you divide the total by the number of values.

*Example: Web Hits*

A program is required which will

- Input the number of hits per day on a web site
- Calculate and display the total and average number of hits

*Sample Values*

| **Day** | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| **Hits** | 859 | 1257 | 724 | 1003 | 982 | 672 | 598 |

Total: 6095 Average: 6095 / 7 = 870.7

*Calculating a Total*

The program will need to add on each input value and add it on to a `total`. Like the `count` variable, it is initialised to zero before the `while` loop. Each time a value is input, the value is added to the `total`

Here's an outline

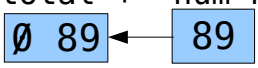
```
count = 0
total = 0
while count < num_values:
    # code to input the value
    value = float(input("Number of hits: "))

    # add the value to the total
    total = total + num_hits

    # increase the counter
    count = count + 1
```

The statement

```
total = total + num_hits
```

can also be implemented using the += operator:

```
total += num_hits
```




This

takes the current value of `total`,
adds on the value of `num_hits`
and then stores the result back in `total`

When `count` reaches `num_values` the while loop stops.

The program then

- displays the total
- calculates and displays the average

```
average = total / num_values
```

*Python Program*

```python
# Input number of values
num_values = int(input("Enter number of values: "))

# initialise count and total variables
count = 0
total = 0

# keep going until the count reaches num_values
while count < num_values:
    num_hits = int(input("Enter number of hits: "))

    # add on to the total
    total += num_hits

    # increase the count by 1
    count += 1

# Display the total
print("Total hits:", total)
print(f"Average: {total/num_values:.1f}")
```

*Sample Output*

```
Enter number of values: 7

Enter number of hits: 859

Enter number of hits: 1257

Enter number of hits: 724

Enter number of hits: 1003

Enter number of hits: 982

Enter number of hits: 672

Enter number of hits: 598
Total hits: 6095
Average: 870.7
```

*How it Works*

Before the loop starts, the `count` and `total` variables are initialised to zero. The counting loop repeats while the `count` variable is less than 7. Each time the loop body executes, it inputs a value, adds it onto the `total` and increase the count by 1. When the `count` reaches 7, the 7 values have been processed, and the `while` loop terminates. The program then displays the `total`, and calculates and displays the `average`.

| count | total | Check: count < 7 | Input | Add to total | Add 1 to count |
|---|---|---|---|---|---|
| 0 | 0 | TRUE | 859 | 859 | 1 |
| 1 | 859 | TRUE | 1257 | 2116 | 2 |
| 2 | 2116 | TRUE | 724 | 2840 | 3 |
| 3 | 2840 | TRUE | 1003 | 3843 | 4 |
| 4 | 3843 | TRUE | 982 | 4825 | 5 |
| 5 | 4825 | TRUE | 672 | 5497 | 6 |
| 6 | 5497 | TRUE | 598 | 6095 | 7 |
| 7 | 6095 | FALSE | | | |

## Maximum and Minimum Algorithms

Suppose the webmin wanted to find out the highest number of hits per day, the maximum value.

With a small number of values, it's easy for a person to do: scan the list and pick out the largest value:

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Hits | 859 | 1257 | 724 | 1003 | 982 | 672 | 598 |

But a computer needs clear instructions how to do it. The approach is as follows:

- Set the maximum value to zero
- Input each value
- Compare it to the maximum: If it is bigger than the maximum, then remember it as the new maximum
- Repeat until all values have been entered

| value | hits | maximum | |
|---|---|---|---|
| | | 0 | starting maximum |
| 1 | 859 | 859 | new maximum |
| 2 | 1257 | 1257 | new maximum |
| 3 | 724 | 1257 | |
| 4 | 1003 | 1257 | |
| 5 | 982 | 1257 | |
| 6 | 672 | 1257 | |
| 7 | 598 | 1257 | |

Here's the pseudo-code use a counting `while` loop:

```
Input number of value
count = 0
max   = 0

while count < number of values
    Input hits
    If hits > maximum
        maximum = hits
        count = count + 1

Print maximum
```

Here's the Python Program:

```
# Input number of values
num_values = int(input("Enter number of values: "))

# initialise count and maximum variables
count = 0
maximum = 0

# keep going until the count reaches num_values
while count < num_values:
    num_hits = int(input("Enter number of hits: "))

    # if the current value is bigger than the maximum so far
    if num_hits > maximum:
        # it becomes the new maximum
        maximum = num_hits

    # increase the count by 1
    count += 1

# Display the maximum
print()
print(f"Maximum hits per day: {maximum}")
```

*Sample Output*

```
Enter number of values: 7

Enter number of hits: 859

Enter number of hits: 1257

Enter number of hits: 724

Enter number of hits: 1003

Enter number of hits: 982

Enter number of hits: 672

Enter number of hits: 598

Maximum hits per day: 1257
```

*How it Works*

First input the number of values, e.g. "How many values will be entered? " The while loop will need to count up to this number. The variable `count` keeps track of how many values have been input. It starts at 0. The maximum also starts at 0.

Because each value is greater than 0, this means that the first value entered will become the first value for the maximum. The while loop checks if the value of `count` is less than number of values. If it is, the statements in the while loop are executed. In this case, the next value is input and compared to the maximum using:

```
if hits > maximum
```

If the new value is greater than the current maximum, then it becomes the new maximum

```
maximum = hits
```

Before the end of the while loop, the count variable is increased by 1, using

```
count = count + 1
```

and the while loop again checks if the value of count is less than number of values. When the value of `count` reaches the number of values, the while loop stops and the maximum is displayed.

| **count** | **max** | **count < number of values?** | **hits** | **hits > max?** | **max = hits** | **count= count+1** |
|---|---|---|---|---|---|---|
| 0 | 0 | TRUE | 859 | TRUE | 859 | 1 |
| 1 | 859 | TRUE | 1257 | TRUE | 1257 | 2 |
| 2 | 1257 | TRUE | 724 | FALSE | | 3 |
| 3 | 1257 | TRUE | 1003 | FALSE | | 4 |
| 4 | 1257 | TRUE | 982 | FALSE | | 5 |
| 5 | 1257 | TRUE | 672 | FALSE | | 6 |
| 6 | 1257 | TRUE | 598 | FALSE | | 7 |
| 7 | 1257 | FALSE | | | | |

*Finding the Minimum*
Suppose the web admininstrator wanted to find out the lowest number of hits per day, the minimum value.

With a small number of values, that's easy for a person to do, scan the list and pick the smallest value:

| Day | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Hits | 859 | 1257 | 724 | 1003 | 982 | 672 | 598 |

But a computer needs clear instructions how to do it.

The approach is not as simple as the one used to find the maximum, where you start with 0 and look for a larger value. If all the values are greater than zero, then you can start with a maximum of 0, and the first value will become the first maximum. You can't do this with the minimum, i.e. you can't start with 0 and look for a smaller value.

```
How many values to enter? 7
Enter number of hits 859
Enter number of hits 1257
Enter number of hits 724
Enter number of hits 1003
Enter number of hits 982
Enter number of hits 672
Enter number of hits 598
Minimum hits per day 0
```

The minimum starts at zero, and because all the values are greater than 0, a smaller value isn't possible – the minimum stays at 0, which is wrong.

The required approach is as follows:

- Input each value
- If it is the first value remember it as the minimum
- Otherwise compare it to the minimum: If it is smaller than the minimum, then remember it as the new minimum
- Repeat until all values have been entered

| value | hits | minimum | |
|---|---|---|---|
| 1 | 859 | 859 | starting minimum |
| 2 | 1257 | 859 | |
| 3 | 724 | 724 | new minimum |
| 4 | 1003 | 724 | |
| 5 | 982 | 724 | |
| 6 | 672 | 672 | new minimum |
| 7 | 598 | 598 | new minimum |

Here's the pseudo-code using a counting `while` loop:

```
Input number of values
count = 0

while count < number of values
    Input hits
    If count == 0
        minimum = hits      Store the first one as the minimum
    Else If hits < minimum
        minimum = hits      Store this one as the new minimum

     count = count + 1

Print minimum
```

*Python Program*

```python
# Input number of values
num_values = int(input("Enter number of values: "))

# initialise count variable
count = 0

# keep going until the count reaches num_values
while count < num_values:
    num_hits = int(input("Enter number of hits: "))

    # if this is the first value
    if count == 0:
        # then set it as the minimum so far
        minimum = num_hits

    # ptherwise, if the current value is smaller than the minimum so far
    elif num_hits < minimum:
        # it becomes the new minimum
        minimum = num_hits

    # increase the count by 1
    count += 1

# Display the minimum
print()
print(f"Minimum hits per day: {minimum}")
```

*Sample Output*

```
Enter number of values: 7

Enter number of hits: 859

Enter number of hits: 1257

Enter number of hits: 724

Enter number of hits: 1003

Enter number of hits: 982

Enter number of hits: 672

Enter number of hits: 598

Minimum hits per day: 598
```

*How it Works*

First input the number of values, e.g. "How many values will be entered? " The while loop will need to count up to this number. The variable `count` keeps track of how many values have been input. It starts at 0.

The while loop checks if the value of `count` is less than number of values. If it is, then the actions in the while loop are performed. In this case, the next value is input. A check is made to see if this is the first value

```
If count == 0
```

in which case this value is stored as the minimum (so far):

```
minimum = hits
```

Otherwise, it's not the first value, so the value is compared to the current minimum using:

```
if hits < minimum
```

If the new value is smaller than the current minimum, then it becomes the new minimum

```
minimum = hits
```

Before the end of the while loop, the count variable is increased by 1, using

```
count = count + 1
```

and the while loop again checks if the value of count is less than number of days.

When the value of count reaches number of days, the while loop stops and the minimum is displayed.

| count | min | count < num_values? | hits | count =0? | min = hits | hits < min? | min = hits | count= count+1 |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | TRUE | 859 | TRUE | 859 | | | 1 |
| 1 | 859 | TRUE | 1257 | FALSE | | FALSE | | 2 |
| 2 | 859 | TRUE | 724 | FALSE | | TRUE | 724 | 3 |
| 3 | 724 | TRUE | 1003 | FALSE | | FALSE | | 4 |
| 4 | 724 | TRUE | 982 | FALSE | | FALSE | | 5 |
| 5 | 724 | TRUE | 672 | FALSE | | TRUE | 672 | 6 |
| 6 | 672 | TRUE | 598 | FALSE | | TRUE | 598 | 7 |
| 7 | 598 | FALSE | | | | | | |

*Combining the Algorithms*
You may want to combine the algorithms for total, maximum and minimum in one.

Here's the code before the while loop:

```
# Input number of values
num_values = int(input("Enter number of values: "))

# initialise count and total variables
count = total = 0
```

Notice how both count and total variables are initialised to zero on the same line.

Here's the while loop:

```
# keep going until the count reaches num_values
while count < num_values:
    num_hits = int(input("Enter number of hits: "))

    # add it on to the total
    total += num_hits

    # if this is the first value
    if count == 0:
        # then set it as the maximum and minimum so far
        maximum = minimum = num_hits
    # otherwise, if the current value is smaller than the minimum so far
    elif num_hits < minimum:
        # it becomes the new minimum
        minimum = num_hits
    # otherwise, if the current value is bigger than the maximum so far
    elif num_hits > maximum:
        # it becomes the new maximum
        maximum = num_hits

    # increase the count by 1
    count += 1
```

Finally, after the while loop, the results are displayed:

```
# Display the total, average, maximum and minimum
print()
print(f"Total hits {total}")
print(f"Average: {total/num_values:.1f}")
print(f"Maximum hits per day: {maximum}")
print(f"Minimum hits per day: {minimum}")
```

*Sample Output*

```
Enter number of values: 7

Enter number of hits: 859

Enter number of hits: 1257

Enter number of hits: 724

Enter number of hits: 1003

Enter number of hits: 982

Enter number of hits: 672

Enter number of hits: 598

Total hits 6095
Average: 870.7
Maximum hits per day: 1257
Minimum hits per day: 598
```

## Python's `for` Loop

Python's `for` loop is another *repetition* control structure. It is used to repeat a block of code. Unlike for loops in other popular programming languages, Python's for loop is not a *counting loop*. Instead it is used to process each item in a sequence, one at a time. Whenever you see a for loop in Python, you should think ***"for each"***

The general syntax of the `for` loop is:

```
for value in sequence:
    statements
```

must end with :

indented

The `for` loop processes each item in the *`sequence`* of values, one at a time. You should read it as:

***"for each value in the sequence"***

The `for` loop line ends with a colon, and the statements associated with the `for` loop are indented. The variable *`value`* after the `for` is called the *loop index*. It is assigned each value in *`sequence`*, one at a time.

## String Processing in a `for` loop

Because a string is a sequence of characters

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| h | e | l | l | o | | w | o | r | l | d |

you can iterate over it with a `for` loop:

```
for character in message:
    print(character)
```

You should read the for loop statement as ***"for each character in the message"***

```
h
e
l
l
o

w
o
r
l
d
```

*How it Works*

This `for` loop takes each character in the string stored in `message` one at a time and assigns each one to the `character` variable.

```
for character in message:
    print(character)
```

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| h | e | l | l | o | | w | o | r | l | d |

The statement in the `for` loop is then executed: in this case, a `print` statement, which prints each character on a separate line.

```
Enter message: Hello World
H
e
l
l
o

W
o
r
l
d
```

If you wish, you can print each character on the same line, separated by a space. This is done by passing the argument `end` set to a single space `" "`

```
for character in message:
    print(character, end=" ")
```

This overrides the `print` function's default, which is to end each output with a newline.

```
Enter message: Hello World
H e l l o   W o r l d
```

**Application: Validate Password**

A program is required to check that a password includes at least one of each the following:

- uppercase letters
- lowercase letters
- digits
- special (non-alphanumeric) characters.

*Examples*

| Password | Valid? | Reason |
|---|---|---|
| &Now4Something | Y | |
| Secret123 | N | No special character |
| OpenSesame! | N | No digit |
| open_sesame123 | N | No uppercase character |

*Approach*

4 boolean ("flag") variables are used to check if a character from each of the 4 categories was found.

They are initialised to `False`

| category | found? |
|---|---|
| lowercase | X |
| uppercase | X |
| digits | X |
| special | X |

| category | found? |
|---|---|
| lowercase | Y |
| uppercase | Y |
| digits | Y |
| special | Y |

A password is valid only if every one of the flag variables is `True`.

*Examples*

| Password | upper | lower | digit | special | valid |
|---|---|---|---|---|---|
| &Now4Something | Y | Y | Y | Y | Y |
| Secret123 | Y | Y | Y | N | N |
| OpenSesame! | Y | Y | N | Y | N |
| open_sesame123 | N | Y | Y | Y | N |

*Python Program*

```python
# Program to check a strong password
password = input("Enter password: ")

# initialise flag variables
lowercase = False
uppercase = False
digit = False
special = False

# check each character in the password
for character in password:
    if character.islower():
        lowercase = True
    elif character.isupper():
        uppercase = True
    elif character.isdigit():
        digit = True
    elif not character.isalnum():
        special = True

# check if all flags are True
if lowercase and uppercase and digit and special:
    print("Valid password.")
else:
    print("Invalid password.")
    print("Include upper and lowercase letters, digits and special characters.")
```

*Sample Output*
The following output is from executing the program 4 different times:

```
Enter password: &Now4Something
Valid password.
```

```
Enter password: Secret123
Invalid password.
Include upper and lowercase letters, digits and special characters.
```

```
Enter password: OpenSesame!
Invalid password.
Include upper and lowercase letters, digits and special characters.
```

```
Enter password: open_sesame123
Invalid password.
Include upper and lowercase letters, digits and special characters.
```

*How it Works*

```
# check each character in the password
for character in password:
    if character.islower():
        lowercase = True
    elif character.isupper():
        uppercase = True
    elif character.isdigit():
        digit = True
    elif not character.isalnum():
        special = True
```

The `for` loop

- stores each password character in `character`
- checks if the character is one of the following
  - An uppercase letter
  - A lowercase letter
  - A digit
  - A non-alphanumeric character

  and sets the corresponding flag variable to `True`

**Application: Validate Username**

A program is required to check that a username matches the following criteria:

- cannot contain Admin
- cannot be longer than 15 characters
- contains only letters, digits and/or underscores.

https://help.twitter.com/en/managing-your-account/twitter-username-rules

*Examples*

| Username | Valid? | Reason |
|---|---|---|
| jbloggs | Y | |
| Joe_Bloggs1990 | Y | |
| MyAdmin | N | Contains Admin |
| Oscar_Fingal_OFlahertie_Wills_Wilde | N | Too long |
| Grace_O'Malley | N | Invalid character ' |

*Python Program*

```python
# Program to validate a username
# Example of a for loop to process the characters in a string
username = input("Enter the username: ")

# Check if the username contains Admin
if "Admin" in username:
    print("Invalid: username must not contain Admin")
# Check if username is too long
elif len(username) > 15:
    print("Invalid: exceeds 15 characters")
# Check if it contains an invalid character
else:
    # process each character, one at a time
    for character in username:
        # if the character is invalid
        if not character.isalpha() and not character.isdigit() and character != '_':
            print("Invalid character: ", character)
            break # exit loop
    # reached end of username
    else:
        print(username, "is valid")
```

*Sample Output*

```
Enter the username: SystemAdmin
Invalid: username must not contain Admin
```

```
Enter the username: Oscar_Fingal_Wilde
Invalid: exceeds 15 characters
```

```
Enter the username: Grace_O'Malley
Invalid character:  '
```

```
Enter the username: jbloggs
jbloggs is valid
```

```
Enter the username: Joe_Bloggs1990
Joe_Bloggs1990 is valid
```

*How it Works*

The `for` loop

- stores each username character in `character`
- checks if the character is invalid:
  - Not a letter
  - Not a digit
  - Not an underscore

  if so, a message is displayed and the loop stops

Just like with a `while` loop, you can add an `else` block to a `for` loop. The `else` block executes if the `for` loop terminates by reaching the end of the sequence. In this case, it means that there are no invalid characters, and therefore the username is valid.

**Creating Strong Passwords: The *Phrase* Approach**

One way to create a strong password involves selecting an easily remembered phrase from a book or film, or the line of a song, and selecting the first or last letter from each word.

| | |
|---|---|
| Example: | In a hole in the ground there lived a hobbit. |
| produces: | Iahitgtlah |

The password is made stronger by adding a digit and a non-alphanumeric character: **Iahitgtlah3<**

Part of the algorithm is:

```
For each word in the line
    Get the first letter of the word
    Add to password
```

To do this in Python, you need to use the string method `split()`, which splits a string into a list of substrings.

Because you have a list (a type of sequence), you can iterate through it using a `for` loop:

```
In [2]: line = "In a hole in the ground there lived a hobbit"

In [3]: line.split()
Out[3]: ['In', 'a', 'hole', 'in', 'the', 'ground', 'there', 'lived', 'a', 'hobbit']
```

```
for word in line.split():
    print(word)
```

e.g.

```
In [4]: for word in line.split():
   ...:     print(word)
In
a
hole
in
the
ground
there
lived
a
hobbit
```

*Python Program*

```python
# Program to generate a strong password using a line of text
from string import digits, punctuation
from random import choice

# input a line of text
line = input("Input a line of text: ")

# set the password to an empty string
password = ""

# for each word in the line
for line in line.split():
    # get the first letter of the word and add letter to password
    password += line[0]

# Capitalise the password
password = password.capitalize()

# randomly select a special character and add it to the password
password += choice(punctuation)

# randomly select a digit and add it to the password
password += choice(digits)

# print password
print('Password is', password)
```

*Sample Output*

```
Input a line of text: In a hole in the ground there lived a hobbit
Password is Iahitgtlah}8
```

*How it Works*

The `password` is initialised as an empty string “”

The `for` loop splits the line into a sequence of words

- processes each `word`, one at a time
- takes the first character of the `word[0]`
- adds it to the `password`

```
In
a
hole
in
the
ground
there
lived
a
hobbit
```

The remaining lines are to

- capitalise the password
- add a random digit
- add a random non-alphanumeric character
- display the password

```
Iahitgtlah
```

### Using `for` as a Counting Loop

A `for` loop *can* be used as a counting loop, for example:

```
In [7]: for i in range(10):
   ...:     print(i)
0
1
2
3
4
5
6
7
8
9
```

*How it Works*

`range(n)` returns an object which provides a sequence from 0 to `n-1`. The `for` loop iterates through the values in the sequence, one at a time, storing each value in the loop index variable `i`.

In this case, `n` is 10, so the `range` object provides the sequence: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

```
                0, 1, 2, 3, 4, 5, 6, 7, 8, 9
for i in range(10):
    print(i)
```

More generally, the syntax of range is: `range(start, stop, step)`

where `start` represents the first value in the sequence
`stop` represents the end point (which will not be included)
and `step` represents the change (increase or decrease) between the values in the sequence.
`step` is optional, and defaults to 1.

For example:

```
In [4]: for i in range(1,20,2):
   ...:     print(i, end=" ")
1 3 5 7 9 11 13 15 17 19
```

`range(1,20,2)` creates a `range` object which provides a sequence starting at 1, increasing by 2 each time, until it reaches 20.

In this case, the output appears on one line, by using `end=""` to replace `print` 's newline character with a space.

*Python Program: Web Hits*

The following program uses a counting `for` loop to calculate the total hits on a web site. The user inputs the number of values, and this is used as the argument for `range`, i.e. `range(num_values)`.

```
# Program to calculate total and average web hits per day
# Counting for Loop

# Input number of values
num_values = int(input("Enter number of values: "))

# initialise total variable
total = 0

# keep going until i reaches num_values
for i in range(num_values):
    num_hits = int(input("Enter number of hits: "))

    # add on to the total
    total += num_hits

# Display the total
print("Total hits:", total)
print(f"Average: {total/num_values:.1f}")
```

*Sample Output*

```
Enter number of values: 7

Enter number of hits: 859

Enter number of hits: 1257

Enter number of hits: 724

Enter number of hits: 1003

Enter number of hits: 982

Enter number of hits: 672

Enter number of hits: 598
Total hits: 6095
Average: 870.7
```

*Python Program*

Similarly, you can use a counting `for` loop to display each character in a string:

- `len(message)` is the number of characters in the string (stored in the variable `message`).
- So `range(len(message))` provides a sequence of integers from 0 to `len(message)-1`
- The `for` loop then uses the index to access the individual characters: `message[i]`

```
# Program to display each character in a string
message = input("Enter message: ")

for i in range(len(message)):
    print(i, message[i])
```

*Sample Output*

The output shows each index and the corresponding character at that index.

```
Enter message: Hello, World!
0 H
1 e
2 l
3 l
4 o
5 ,
6
7 W
8 o
9 r
10 l
11 d
12 !
```

### The continue Statement

The continue statement is used to end the current iteration of a for loop or while loop and continues with the next iteration of the loop.

For example, the following code is used to replace each letter in a message with its position in the alphabet, omitting other characters:

```
# Example of a continue statement in a for loop

# Program to replace each letter with its position in the alphabet
# This is a simple substitution cipher: the original message is the "plaintext"
# and the enciphered result is the "ciphertext"

from string import ascii_lowercase #abcdef...xyz

# Input a message
plaintext = input("Enter the message to be enciphered: ")

ciphertext = ""

# for each character in the plaintext, converted to lowercase
for character in plaintext.lower():
    # if the character is not a letter, skip it
    if not character.isalpha():
        continue # skips this character - goes to the for statement

    else:
        # get position of the letter in the alphabet
        index = ascii_lowercase.index(character)

        # add the index and a space onto the ciphertext
        ciphertext += str(index) + " "

# Display the ciphertext
print("The ciphertext is:", ciphertext)
```

This is an example of a *simple substitution cipher*: the original message is the "plaintext" and the enciphered result is the "ciphertext", which consists of each letter in the plaintext substituted by another letter, number or other symbol. In this case, each letter is substituted by its position (index) in the alphabet, i.e. a=0, b=1, c=3, etc.

*Sample Output*

```
Enter the message to be enciphered: Hello, World!
The ciphertext is: 7 4 11 11 14 22 14 17 11 3
```

*How it Works*

The for loop processes each character in the plaintext string. If it is not a letter, then the continue statement skips the rest of the for block and control moves to the next iteration of the **for** loop. If it is a letter, then its index in the alphabet is converted to a string and added on the the ciphertext, along with a space to separate the values.

### The `pass` Statement

The `pass` statement does nothing. It can be used when a statement is required syntactically but the program requires no action.

For example:

```
while True:
    pass  # Wait for keyboard interrupt (Ctrl+C)
```

https://docs.python.org/3/tutorial/controlflow.html

The `pass` is also useful in places where your code will eventually go, but has not been written yet (e.g., in function stubs). It can also be helpful when you have created a code block but it is no longer required, or you want to temporarily exclude it: you can remove/comment out the statements inside the block but let the block remain with a `pass` statement so that it doesn't interfere with other parts of the code.

https://www.tutorialspoint.com/python/python_loop_control.htm