

# Loops

# Loops

Loops are a way to repeat code multiple times.

- They are good way to allow more flexibility in your code.
- They allow you to perform code a different number of times based on what is happening or has happened in your code.

There are two types of Loops. For and While.

- In many situations they can both be used, but most situations work better or are easier to code with one.

Just like conditionals, **indentation is very important**. Everything in a loop must be indented.



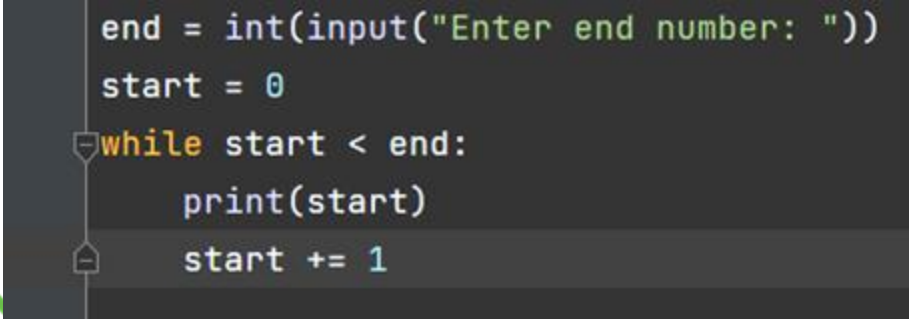
# While Loops

While loops run as long as the given condition returns a True boolean.

Think of it like a repeating if statement.

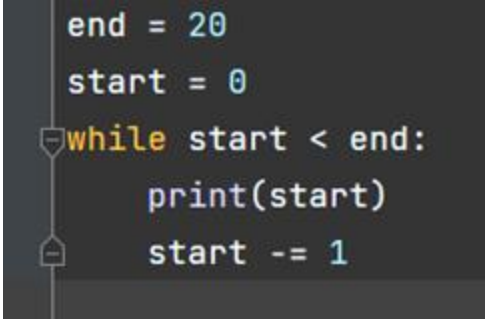
These are great for doing things like making sure someone inputs the correct thing, or an update loop that every second prints text.

Be careful of creating infinite loops.



```
end = int(input("Enter end number: "))
start = 0
while start < end:
    print(start)
    start += 1
```

A code editor window with a dark background. The code is written in a monospaced font with syntax highlighting. The variable 'end' is assigned the value of an integer input from the user. 'start' is initialized to 0. A 'while' loop is defined with the condition 'start < end'. Inside the loop, 'start' is printed and then incremented by 1. A small logo with four colored squares (blue, green, yellow, pink) is visible in the bottom-left corner of the editor window.



```
end = 20
start = 0
while start < end:
    print(start)
    start -= 1
```

A code editor window with a dark background. The code is written in a monospaced font with syntax highlighting. The variable 'end' is assigned the value 20. 'start' is initialized to 0. A 'while' loop is defined with the condition 'start < end'. Inside the loop, 'start' is printed and then decremented by 1.

# Example

Create a while loop that prints every integer from 1 to 10.



# While Loops and User Input

While loops are often paired with user input. The condition for the loop might be what the user needs to enter to stop the loop. The user is prompted for input each time it loops, and something happens based on that input.

This allows you to take user input multiple times without writing multiple lines of `input()`. It also allows the user to control how much input they give.

```
userin = "" # Set userin to an empty string to start out
while userin != "stop": # If the user enters "stop" end the loop
    userin = input("Enter something: ") # Get user input
    print(userin) # Print the user input
    # Usually you would do more than just print it

print("Done collecting user input.")
```



# Exercise

Improve the login system we wrote last class to allow multiple attempts. You're developing a login system for a website. Write a Python program that checks whether the user has entered the correct username and password. Just like before:

- Create two variables called `username` and `password`.
- Prompt the user to enter their username and password.
- Use conditionals and logical operators to check whether the username and password entered by the user match the `username` and `password` variables.
- As long as the username and password are incorrect, print “Incorrect username or password”, and keep asking the user for their username and password.
- If they match, print “Login successful” and end the program.



# For Loops

- For loops are used to iterate through something.
- For loops perform an action on a group of objects.
- For loops can be performed on iterables:
  - **Strings**
  - Lists
  - Tuples
  - Dictionaries
  - Sets
  - ...etc.

```
for letter in "Hello There!":  
    print(letter)
```



# For Loops

A Python keyword, meaning that each item is **in** the collection

```
for item in collection:
```

```
    print(item)
```

The name of an existing variable. This must be an iterable - a collection of items - such as a string, list, etc. defined before the for loop

A temporary variable that doesn't already exist. This steps through every item in collection. You can name it whatever you want. It disappears (goes out of scope) once the for loop ends.





# Example

Write a for loop that loops through a string, counts all the letters, and then print how long the string is.



# Exercise

Take a string from the user. Verify that it's a number.

Write a for loop that adds all the digits together. Then print the total.

## Example:

'14253'

15

Hint: remember to cast to `int()` for each digit in the loop



# Conditionals in Loops

- Adding conditionals to loops is a great way to make a loops code more dynamic.
- They allow us to do different things based on what is happening in the loop.
- They also are an easy way to evaluate if a while loop should end.

```
word = "hello"  
vowels = ['a', 'e', 'i', 'o', 'u']  
for letter in word:  
    if letter in vowels:  
        print('Vowel')  
    else:  
        print(letter)
```



# Exercise

Write some code that will loop through a string and print whether each letter is a vowel or consonant.

## Example:

```
'hello'
```

```
h is a consonant
```

```
e is a vowel
```

```
l is a consonant
```

```
l is a consonant
```

```
o is a vowel
```



## Exercise - Cleaning Strings

You're working on a data analysis project for a company that looks at written text. You're only interested in letters from A-Z because you're analyzing language. However, the data you're given has some values that shouldn't be there.

Write a Python program that takes a string as input from the user, removes anything from the string that isn't a letter, and prints the new string.

You can loop through the string in a for loop, use the `.isalpha()` string method, and remember that strings are immutable, so you will have to build a new string from scratch using string concatenation.

