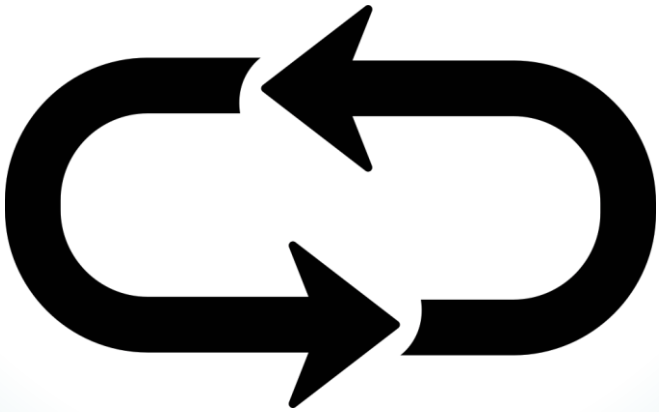
# Computer Science
# Programming Fundamentals

## Loops (While)



# Loops (While) Intro

As long as you are alive, your heart will beat. As long as the light is on, it will use a little but of electricity. Many things in life follow this pattern. It can also be useful to us in coding!

Sometimes we want a part of the code to keep running, again and again, until we are ready for it to stop. Let's see how while loops can help us do this!

# Loops (While) pt 1

- Run the example:

```
1  import time
2
3  alive = True
4
5  while alive:
6    print("eat")
7    time.sleep(0.3)
8    print("sleep")
9    time.sleep(0.3)
10   print("code")
11   time.sleep(0.3)
```

In this code, we import time so we can use a "wait" function called `sleep()`.

Then, we make a new variable called `alive` and set it to `True`.

When you make a variable equal to True or False, you are making a **boolean** variable. This means it can only have one of two values: `True`, or `False`.

# Loops (While) pt 2

You'll notice that this code will run forever, because `alive` will never be `False` in this code.

- Stop the code by making this change, then running it:

```
alive = False
```

Now, this code does nothing at all. Why? Because the while loop will not run if `alive = False`.

# Loops (While) pt 3

- Make these changes and run:

```
import time


alive = True
counter = 0


while alive and counter < 2:
  print("eat")
  time.sleep(0.3)
  print("sleep")
  time.sleep(0.3)
  print("code")
  time.sleep(0.3)


  counter = counter + 1
```

# For (While) pt 4

Q: Why does the loop only run two times?

**A:** Because we added a new part to the condition. Now, the loop will only run while `alive` is `True` AND `counter` is less than 2. Since we add one to `counter` on each run, the loop can only run two times before the condition is false.

# Loops (While) pt 5

Here's another way to stop the loop.

- Make these changes and run:

```
import time

alive = True
counter = 0

while alive:
  print("eat")
  time.sleep(0.3)
  print("sleep")
  time.sleep(0.3)
  print("code")
  time.sleep(0.3)

  if counter == 2:
      break

  counter = counter + 1
```

# Loops (While) pt 6

Here, we're telling Python to break, or stop, the loop when `counter` is 2.

Q: Why does it print each thing three times though?

**A:** Look at where we do the test. We test `counter` after we do all of our print statements. If you put the condition in the head of the loop, then the test will happen before it starts each run.

# Loops (While) pt 7

## Activity:

- We'll make a battleship game on a 3x3 grid where you play against the computer. The ship will be 1x1 in size. We will **validate** the user's input to make sure he doesn't write a location on the grid that doesn't exist.


- When the first function is completed, we will design and use a test plan to make sure it knows what to accept and what to reject.

# Loops (While) pt 8

## Example Test Plan:

| Test Number | Test Data | Type of Test Data | Correct result | Actual Result |
|---|---|---|---|---|
| 1 | 0 | **Boundary** | Accept | |
| 2 | | | | |
| 3 | | | Accept | |
| 4 | | **Erroneous** | "Try again" message | |
| | | Erroneous | "Try again" message | |
| | | | "Try again" message | |

The second column shows what data we plan to enter for our test.

The third column shows what kind of data we plan to enter. Will it be good data, bad data, or data that's right on the edge?

The fourth column shows what should happen.

The last column is where we write what actually happened when we did the test.

# Loops (While) pt 9

## 🔎 Explanation:

First, we import random so we can use some random numbers in our program.

On lines 3 through 7, we set up our grid.

On line 8, we randomly place the ship.

Starting on line 10, we make a function to validate the user's input. It takes a message as a parameter. We will give that message to the input function we call inside `validate_input()`.

In that function, we ask for an int version of the user's input.

On line 11, we set `valid` to `False` because we will assume it is not valid until we know that it is.

Starting on line 13, we make a while loop that keeps repeating as long as `valid` is `False`.

In that loop, We test to make sure that what the user entered (`cor`) is greater than or equal to 0, but less than the length of the grid.

If the user wrote a place that's on the grid, Python moves on to the next step. Otherwise, it will tell the user to try again and let the user try again.

Lastly, the function gives back the user's input, which we now know will be valid.

Remember! The first index of a list is 0, so valid numbers will be from 0 to 2. Three is invalid.

# Loops (While) pt 10

## 🔎 Explanation pt 2:

On line 24, we give the player 5 torpedoes.

Starting on line 26, we make a while loop that will repeat forever until we stop it from the inside.

First, the loop prints out the map, which is grid. It prints the grid row by row.

Then, it lets the player pick a place to shoot the torpedo. We validate this input so it doesn't break our program.

After that, we test what the user entered to see if it matches the ship's location. If it does, Python says "It's a hit!" and ends the loop. Otherwise, we just put an X on the grid so the user knows he already tried that place.

Line 40 takes away one torpedo.

Lines 42 - 45 print two messages and break the loop when the player runs out of torpedoes. The messages let the user know he has no more torpedoes and tells him where the ship was hiding.