Learning Objectives - While Loops

- **Explain while loop syntax (especially the whitespace)**
- **Identify the causes of an infinite loop**
- **Describe the `break` statement as it relates to a while loop**

While Loops

While loops differ from for loops in several ways.

While Loop	For Loop
Runs as long as a **condition** is true	Runs a predetermined amount of times
You must declare a counting variable	Counting variable is automatically declared
You must increment the counting variable	The counting variable is already incremented

While Loop Syntax

While loops, just like for loops, use a `:` and indent for all commands that should be repeated. Here is a while loops that prints “Hello” five times.

```
count = 0 # counting variable
while count < 5:
    print("Hello")
    count = count + 1
```

challenge

What happens if you:

- Change the while statement to `while count < 10:`?
- Change the last line of code to `count = count + 2`?
- Change the while statement to `while count < 0:`?

Infinite Loops

Infinite loops are loops that never have a test condition that causes the loop to stop. For example, this is a common mistake:

```
count = 0 # counting variable
while count < 5:
    print("Hello")
```

Since the variable `count` never gets incremented. It remains 0, and 0 will forever be less than 5. So the loop will never stop.

warning

Run the code above to see what happens. Python will eventually stop the loop due to an output limit, but it will take some time before this happens. Since for loops run for a predetermined amount of time, you do not see infinite loops with them.

Why Use a While Loop?

If a while loop does the same thing as a for loop and infinite loops can occur in a while loop, why use them? While loops are useful when you are waiting for a certain event to occur. Imagine you are making a video game. The game should continue until the player loses all of their lives. You don't know how long this will take, so a while loop would be appropriate.

```
player_lives = 3

while player_lives > 0:
    # video game code
    # goes here
```

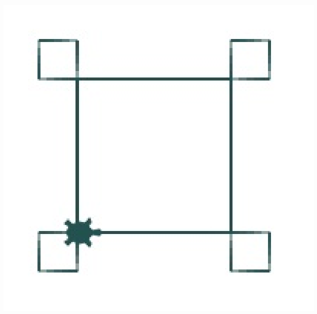
Turtle Coding - While Loop

Instead of a for loop, recreate the images below using while loops. Close the window with the turtle output to stop your program.

▼ **Turtle Graphics Refresher**

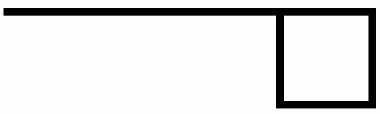
- `t.forward(10)` - Takes a number for the distance traveled
- `t.backward(10)` - Takes a number for the distance traveled
- `t.rt(45)` - Takes a number for degrees turned
- `t.lt(45)` - Takes a number for degrees turned
- `t.color('red')` - Takes a string for the color
- `t.shape('turtle')` - Takes one of the following strings `'turtle'`, `'circle'`, `'square'`, `'arrow'`, `'classic'`, or `'triangle'`.
- `t.pensize(4)` - Takes a positive number
- `t.speed(1)` - Takes a number in the range 0..10

Challenge 1


Turtle Challenge 1

▼ **Hint**

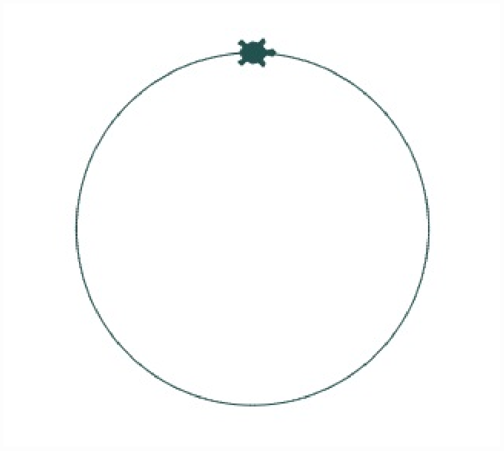
The trick is to find the pattern and then repeat it four times. The pattern is to go forward and then make a smaller square (with right turns) at the end. The pattern should look something like this:



▼ **Hint 2**

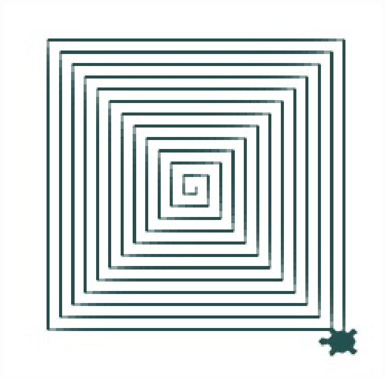
If you are stil stuck, use these lines of code to see if it will help you complete the pattern: `t.forward(80)`, `t.rt(90)`, `t.forward(20)`.

Challenge 2


Turtle Challenge 2

▼ **Hint**
Since a circle has 360 degrees, you will need a loop that repeats 360 times. Be careful about how far the turtle walks. The circle can get very big, very quickly.
▼ **Hint 2**
If you are stil stuck, use these lines of code to see if it will help you complete the pattern: `t.rt(1)`.

Challenge 3


Turtle Challenge 3

▼ **Hint**
The pattern here is to move forward and make a right turn. The trick is, the amount to move forward needs to get bigger as the loop advances. Think of some operators that you can use to make the loop variable get a little bit bigger each iteration.
▼ **Hint 2**
If you are still stuck, multiply the value of `i` by another number to generate the distance the turtle needs to move forward.

Break Statement

Infinite Loops Are Bad, Right?

Well, that depends. If an infinite loop caused because the counting variable isn't incremented, then yes that is a bad thing. But some programmers purposely create a condition that will always evaluate to true. Therefore, the loop will always run. However, a `break` statement is used to stop the loop when a certain condition is true. Copy/paste the code below, and run it several times.

```
import random

while True:
    print("This is an infinite loop")
    rand_num = random.randint(1, 101) # random integer between 1
        and 100
    if rand_num > 75:
        print("The loop has ended")
        break # stop the loop
```

Even though `while True` will always evaluate as a true statement, the loop never becomes infinite because of the `break` statement.

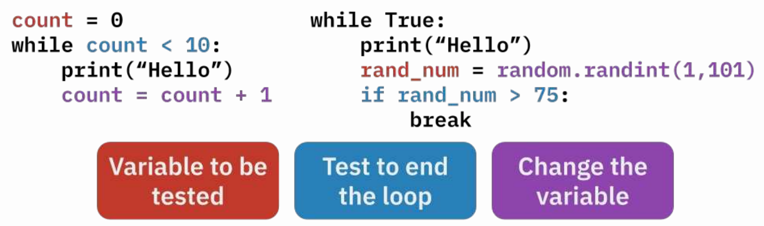
challenge

What happens if you:

- Remove the `break` statement?
- Move the `break` statement to before the `print` statement?

Comparing While Loops

Even though the while loops introduced on the previous page look different than the while loops covered on this page, they both have the same components and do the same thing.



```
count = 0
while count < 10:
    print("Hello")
    count = count + 1
```

```
while True:
    print("Hello")
    rand_num = random.randint(1,101)
    if rand_num > 75:
        break
```

Comparing While Loops

Formative Assessment 1

Formative Assessment 2