Repetition Structures

Chapter 4

Loops

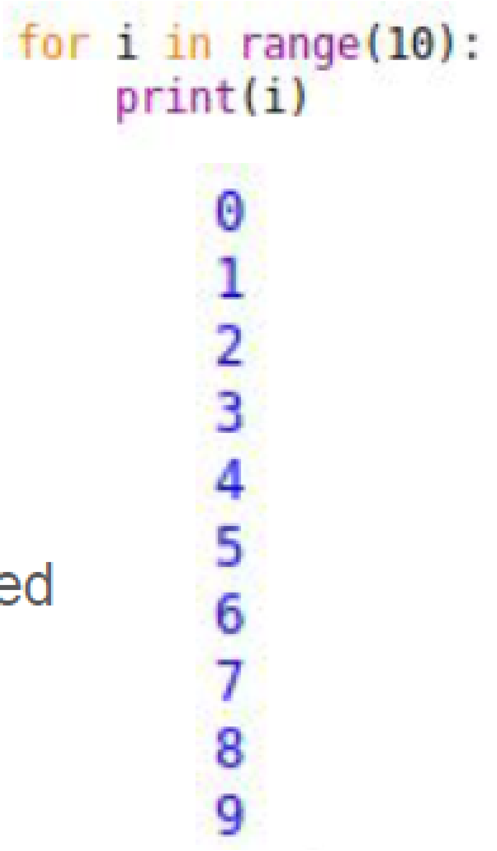
- Loops are a way to repeat segments of code a certain number of times or until a certain condition is met
- Without a loop, there is no way a segment of code will be run more than once
    - (* unless you get into recursion, which often confuses even seasoned programmers)
- Python (and most other languages) has two forms of loops: for loops and while loops
- The loop-body (what you want the loop to do) follows same convention as conditions
    - Whatever is inside the loop must be indented
    - Whatever is outside the loop is unindented

for Loops

- Used to complete an action a certain number of times
  - Can use a variable for the number of times it will run
  - Has to be a predetermined number of times (cannot dynamically determine it)
- Also used to count items
  - More uses for this become apparent once sequences are introduced
- Referred to commonly as “count-controlled loops”

for Loops Continued

- Created by keyword “for”
- Uses Python built-in function “range”
    - Counts start at 0, not 1 (unless stated otherwise)
    - Ranges work on “less than” not “less than or equal to”
    - Each portion of the loop adds 1 (unless stated otherwise)
- i in this example is your “iterator” or the variable assigned to carry your counting value



```
for i in range(10):
    print(i)
```

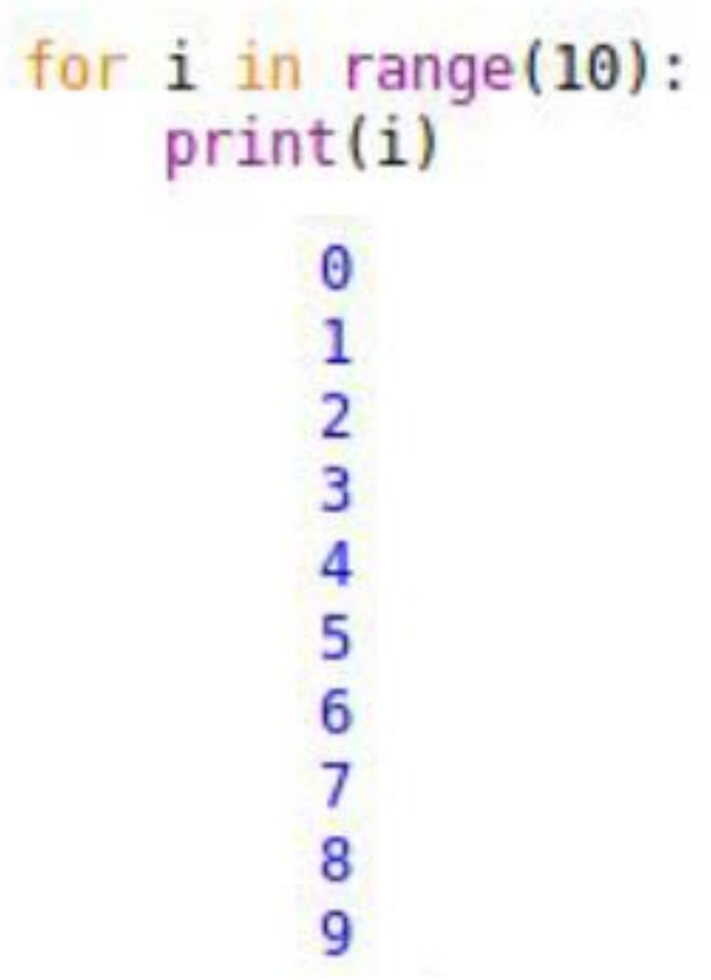
```
0
1
2
3
4
5
6
7
8
9
```

range Function Incrementing

- range() can have 1, 2, or 3 arguments

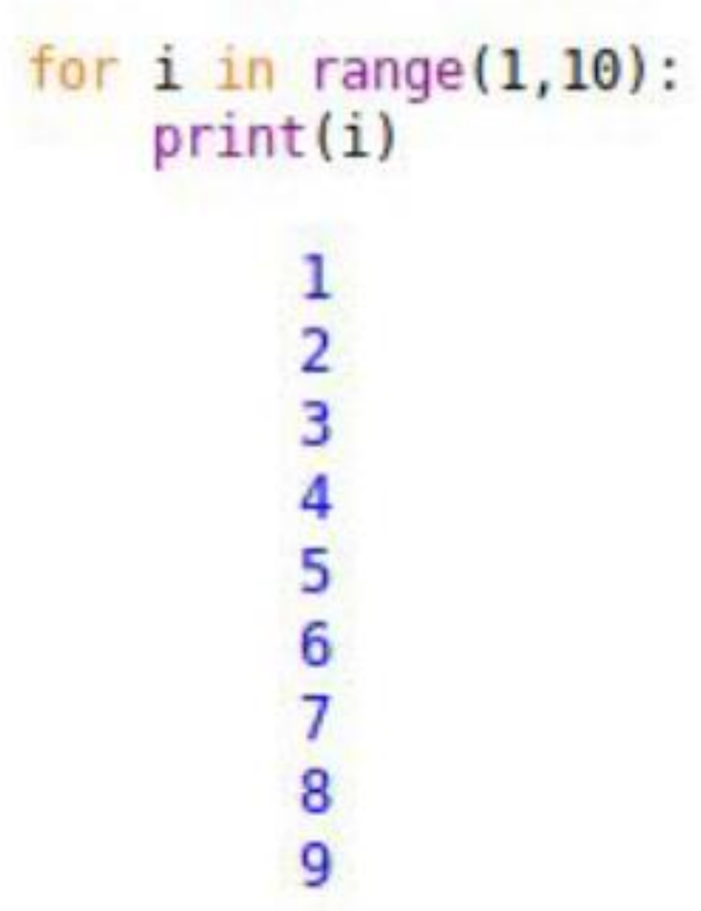
Number of Arguments	Arguments Definition
1 -- ex. range(10)	Count starts at 0, goes to less than the amount specified in first argument, adds one on each pass through the loop
2 -- ex. range(1, 10)	Count starts at first argument (1 in the example), goes to less than the amount specified in second argument, adds one on each pass through the loop
3 -- ex. range(0, 10, 2)	Count starts at first argument (0 in the example), goes to less than the amount specified in second argument, adds the amount in the third argument (2 in the example) on each pass through

range Examples



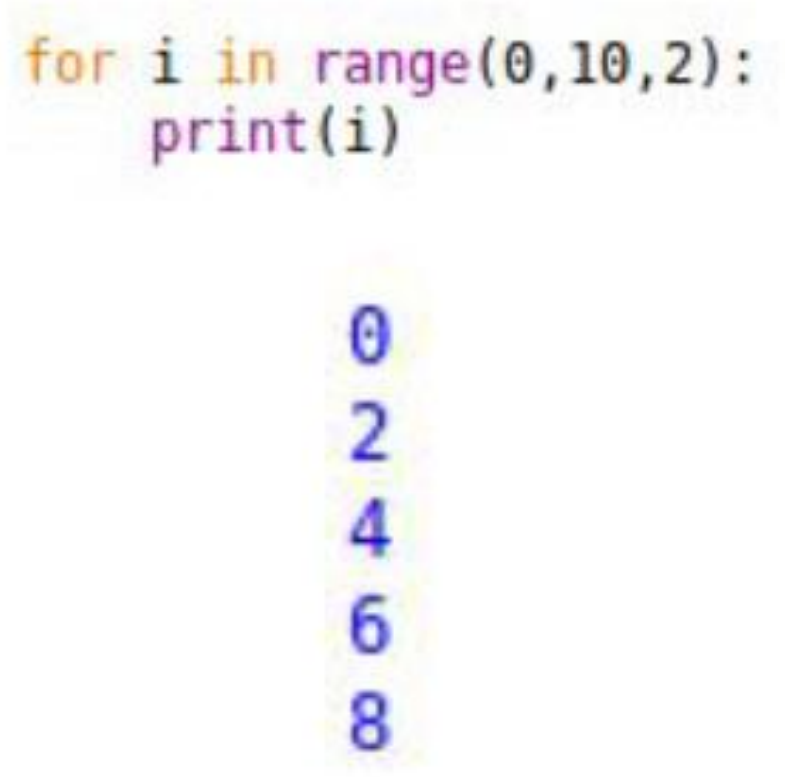
```
for i in range(10):
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9
```



```
for i in range(1,10):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
```

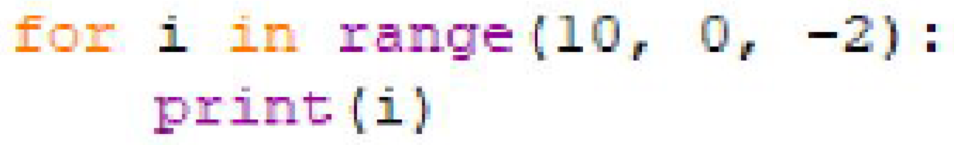


```
for i in range(0,10,2):
    print(i)
```

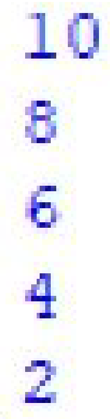
```
0
2
4
6
8
```

range Function Decrementing

- With decrementing, you can only use the 3 argument range version
- You specify the starting point (first argument), the ending point (second argument), and the amount you're subtracting by (third argument)
- Goes to greater than the ending point, not greater than or equal to



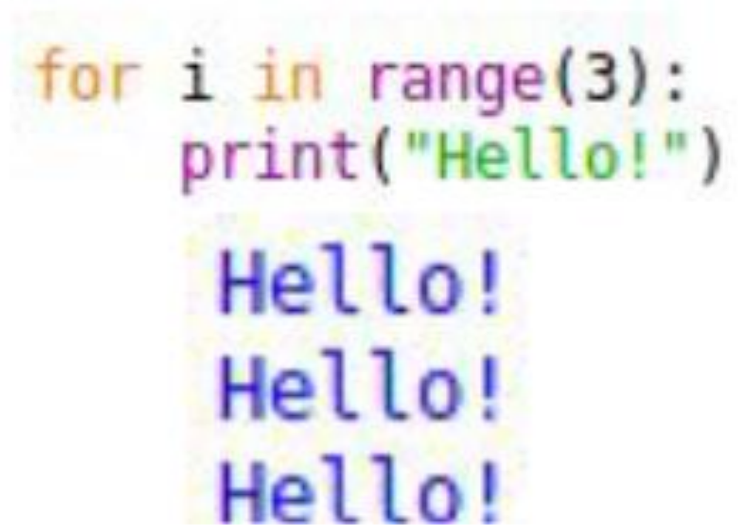
```
for i in range(10, 0, -2):
    print(i)
```



```
10
8
6
4
2
```

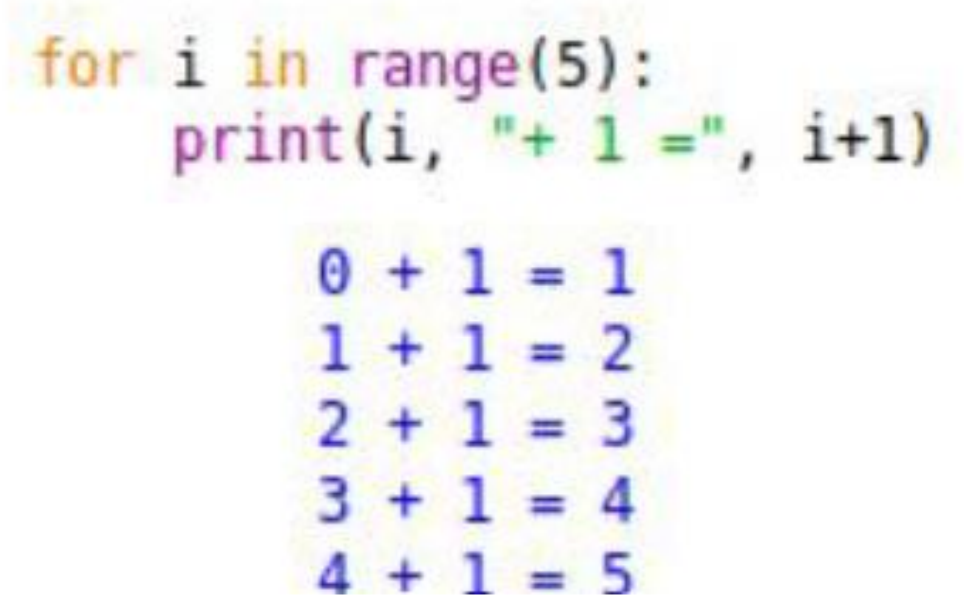
for Loop Notes

- Don't *have* to print out or even use the count in a for-loop
  - Can be used purely to repeat an action a certain amount of times
- Can use the count in calculations, not just print it



```
for i in range(3):
    print("Hello!")
```

```
Hello!
Hello!
Hello!
```

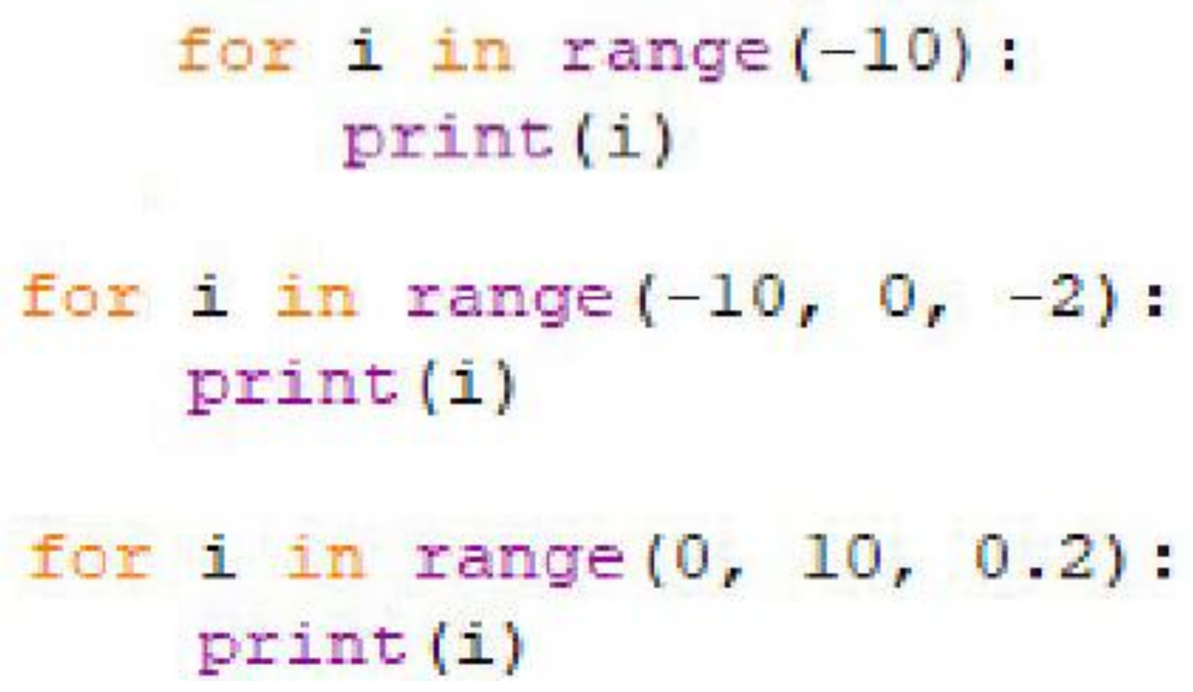


```
for i in range(5):
    print(i, "+ 1 =", i+1)
```

```
0 + 1 = 1
1 + 1 = 2
2 + 1 = 3
3 + 1 = 4
4 + 1 = 5
```

for Loop Notes Continued

- You can have for loops that do nothing
  - Take a look at your range function and see if it's attainable
- The range function only accepts whole numbers (no decimals)



```
for i in range(-10):
    print(i)
```

```
for i in range(-10, 0, -2):
    print(i)
```

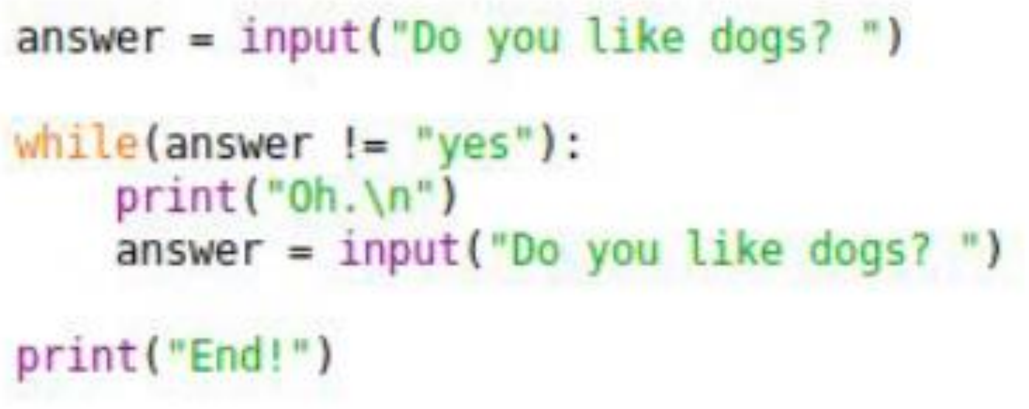
```
for i in range(0, 10, 0.2):
    print(i)
```

while Loops

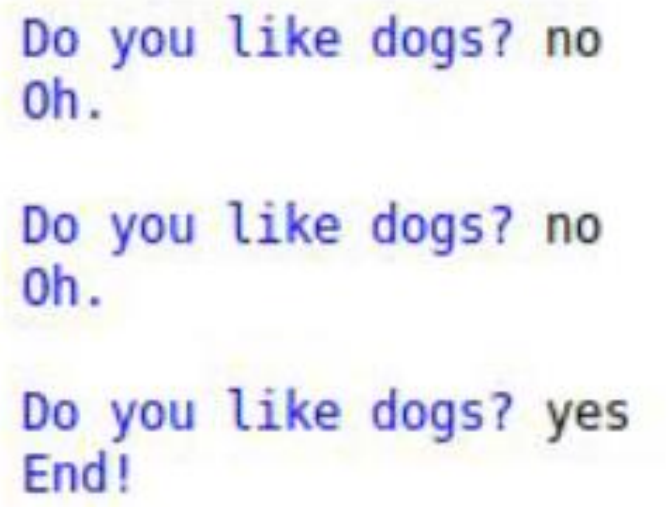
- Used to repeat a segment of code until a certain condition is met
    - Conditions are in the same vein as if-statements
- Can dynamically determine when to end the loop (due to the conditions)
- Can emulate for-loops, but they aren’t as efficient
- Referred to as “condition-controlled loops”

while Loops Continued

- Created by keyword “while”
- Uses conditions, much like if-statements do
- Have to ***change*** the variable being compared or you’ll create an infinite loop (a loop that does not end)
- Code after the while-loop does not execute until the conditions of the while loop have been met



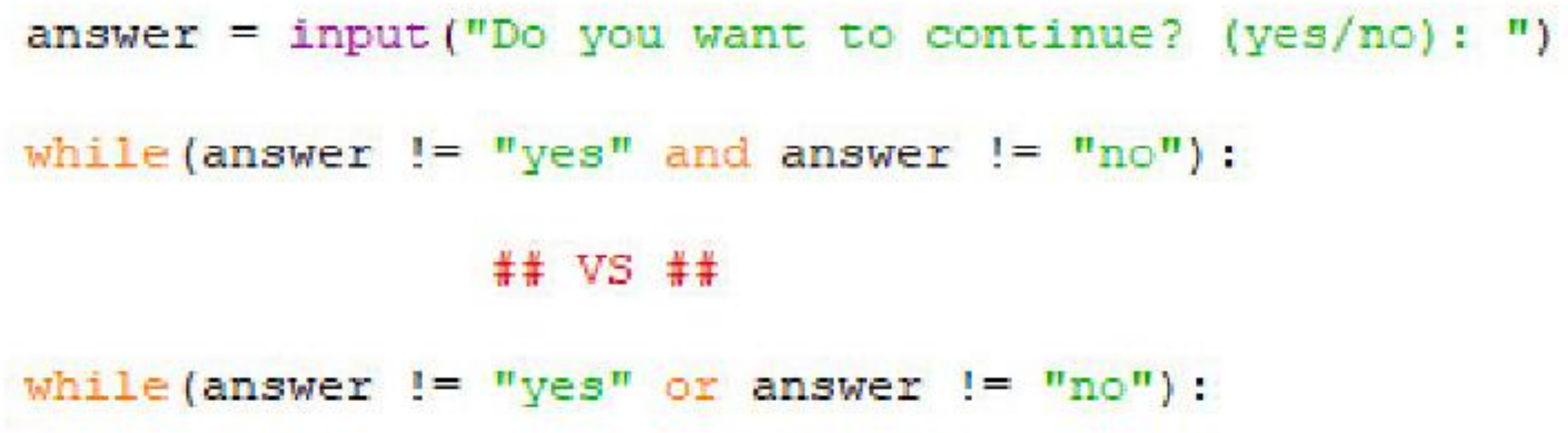
```
answer = input("Do you like dogs? ")

while(answer != "yes"):
    print("Oh.\n")
    answer = input("Do you like dogs? ")

print("End!")
```



```
Do you like dogs? no
Oh.

Do you like dogs? no
Oh.

Do you like dogs? yes
End!
```

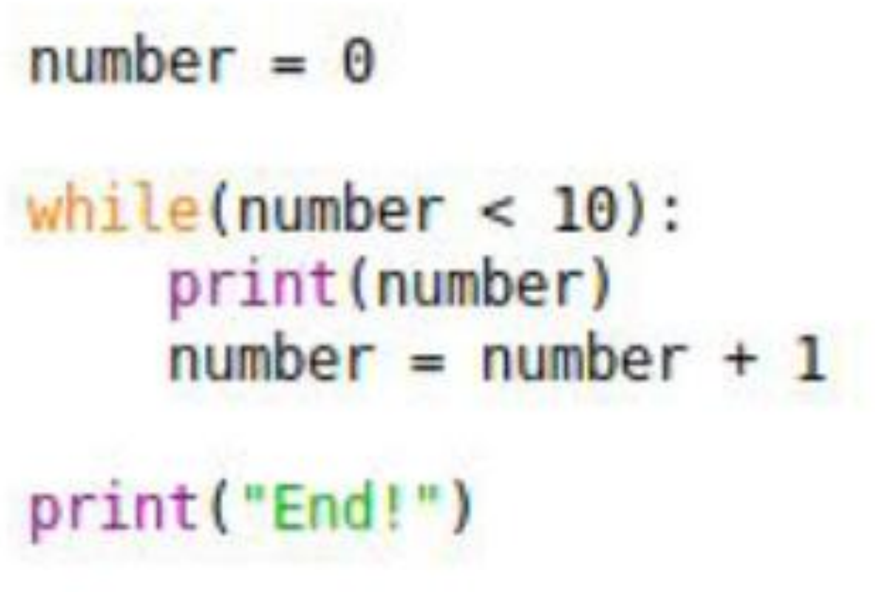
while Loops Continued

- Like if-statements, you can have compound conditions
- Pay attention to what and/or combination you're doing
  - This could affect whether your loop is doing what it should be
  - It could unintentionally create an infinite loop

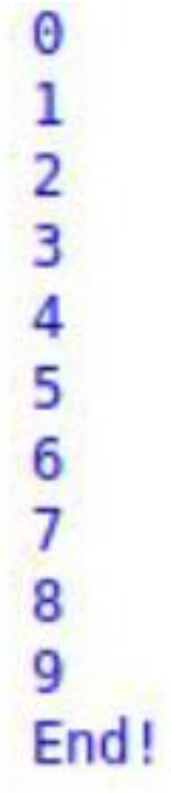


```
answer = input("Do you want to continue? (yes/no): ")

while(answer != "yes" and answer != "no"):

                ## VS ##

while(answer != "yes" or answer != "no"):
```

Simulated for Loop



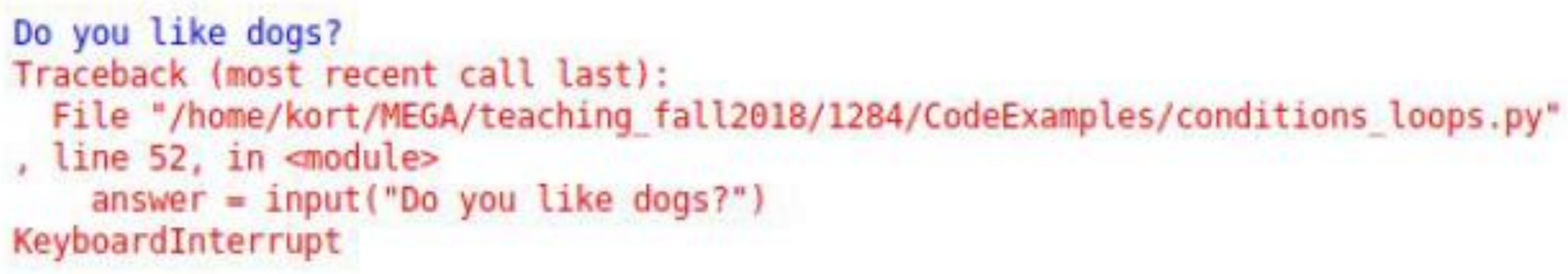
```python
number = 0

while(number < 10):
    print(number)
    number = number + 1

print("End!")
```



```
0
1
2
3
4
5
6
7
8
9
End!
```

while Loop Notes

- If in an infinite loop, you can “ctrl-C” to break out of the loop
- If you see yourself in an infinite loop, check:
  - Am I changing the thing I’m comparing?
  - Am I checking the right thing?
  - Is it a compound condition? Am I using and/or correctly?
  - Is my logic correct?



```
Do you like dogs?
Traceback (most recent call last):
  File "/home/kort/MEGA/teaching_fall2018/1284/CodeExamples/conditions_loops.py"
, line 52, in <module>
    answer = input("Do you like dogs?")
KeyboardInterrupt
```