ShiP.py

Learn to Py while Shelter-in-Place

L3: Repetitions (Looping)

A volunteering educational initiative during COVID-19

ShiP Crew

JD

Teddy

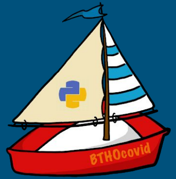
Chinmay

Pratik

Siddharth

Umang

Waseem



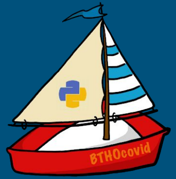
A volunteering educational initiative during COVID-19

Topics

PHASE I: Foundations

All times are in CDT (GMT-5)

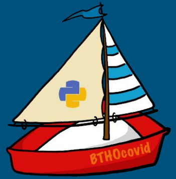
1.	Variables, Expressions, Simple I/O	Sat, April 18 (11 am-12 noon)	☀️
2.	Boolean Decisions (branching)	Wed, April 22 (9 pm-10 pm)	🌙
3.	Repetitions (loops)	Sat, April 25 (11 am-12 noon)	☀️
4.	Collective Data Structures	Wed, April 29 (9 pm-10 pm)	🌙
5.	Functions	Sat, May 02 (11 am-12 noon)	☀️
6.	File I/O	Wed, May 06 (9 pm-10 pm)	🌙
7.	X	Sat, May 09 (11 am-12 noon)	☀️



Lecture 3

AGENDA

- Repetitions/ Loops
- while loop
- Iterables and Iterators
- for loop
- Nested loops
- break & continue

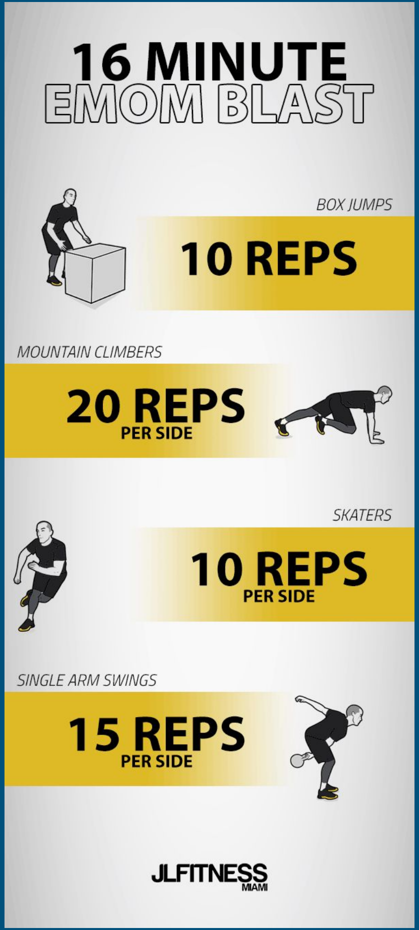


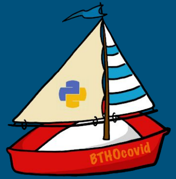
Repetitions

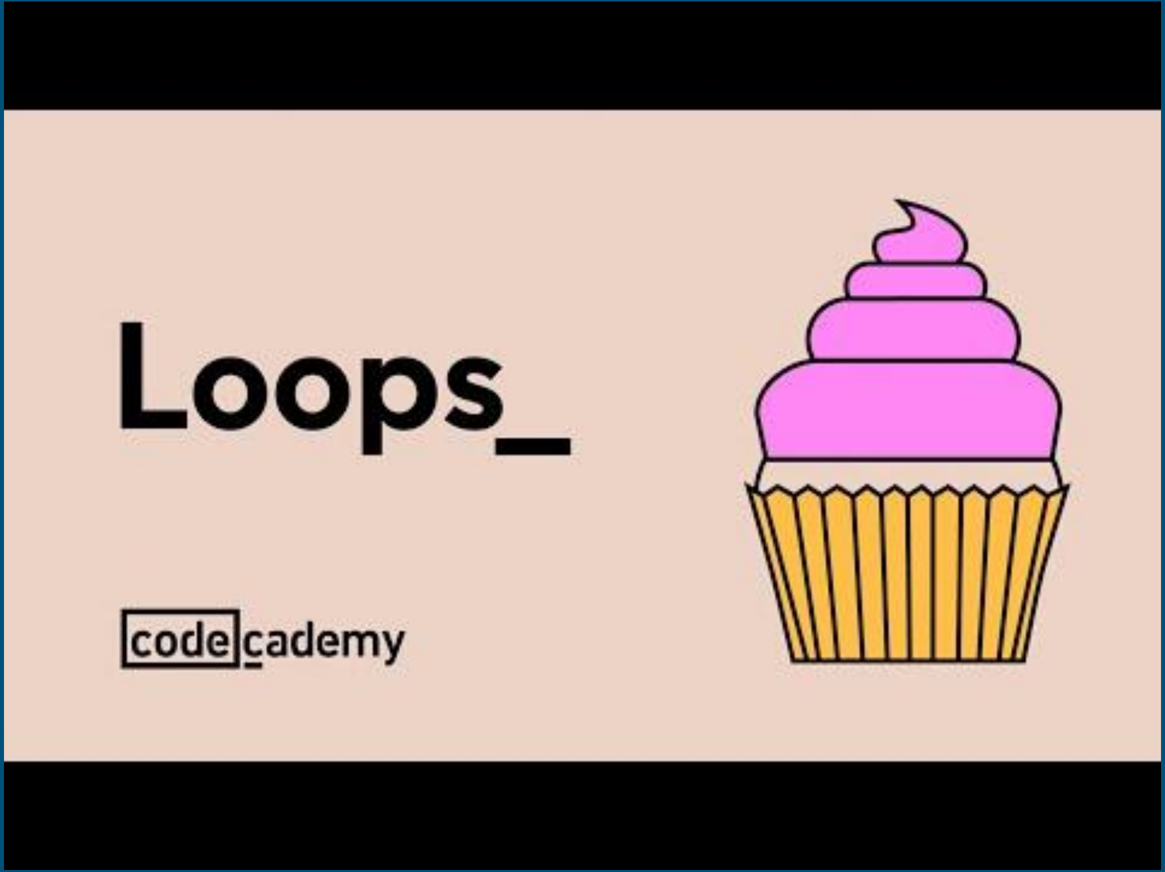
When step(s) needs to be done multiple times

Consider eating a bowl of cereal

1. put cereal in bowl
2. add milk to cereal
3. spoon cereal and milk into mouth
4. repeat step 3 until all cereal and milk is eaten
5. rinse bowl and spoon

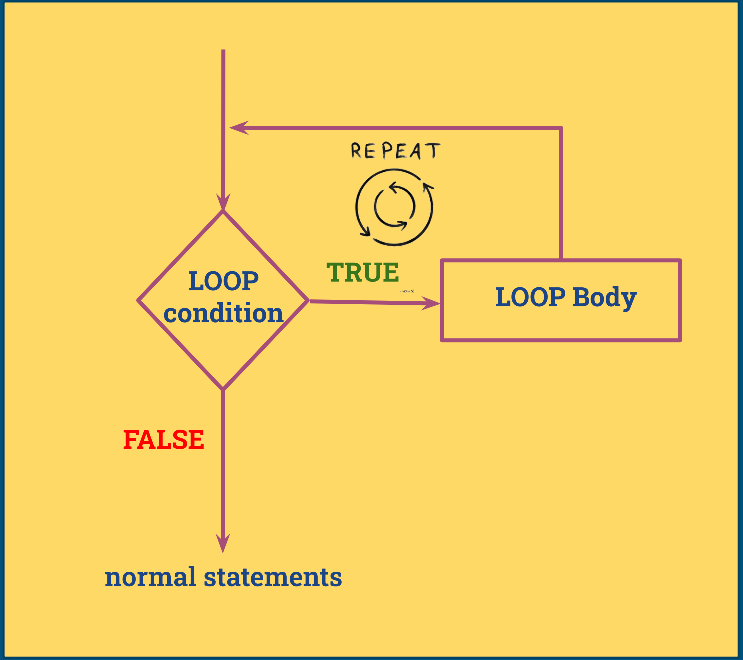


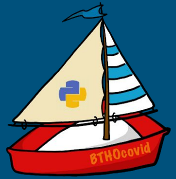



Loops_
codecademy

Loops

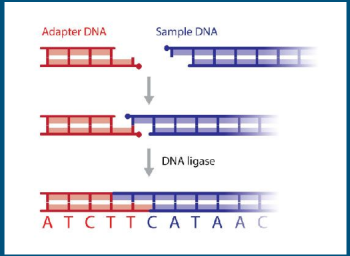
- Used in situations where a piece of code is repeated until loop condition is violated
- Condition- to test if we need to repeat the code or stop
- Loop Body- a piece of repeatable code
- Program continues normally after loop



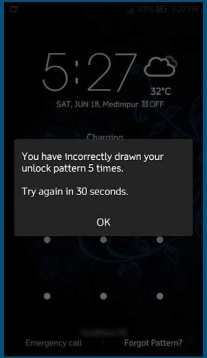


Example Scenarios - Loops

process transaction after transaction until you acknowledge that you have no more to do



Joining/ trimming the fixed length adapter sequence from multiple DNA sequences



You have incorrectly drawn your unlock pattern 5 times.

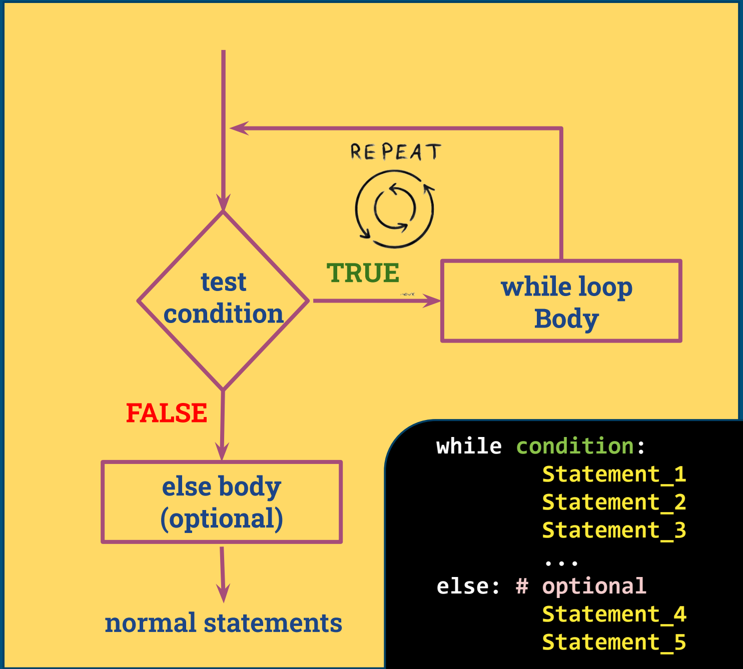
Try again in 30 seconds.

allows user to unlock the mobile with 5 password attempts. After that it waits for 30 seconds and restart the process

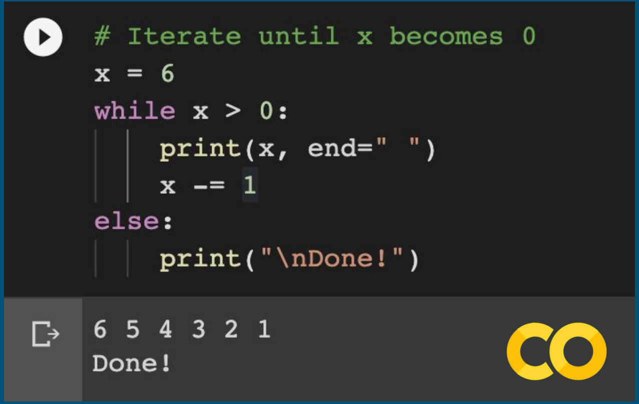
Favorite song on a repeat

Fixed number of rounds for each ride

while loop

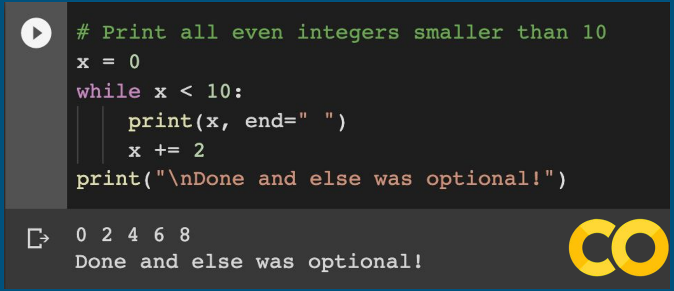


```
while condition:
        Statement_1
        Statement_2
        Statement_3
        ...
else: # optional
        Statement_4
        Statement_5
        ...
# normal statements follow
```



```
# Iterate until x becomes 0
x = 6
while x > 0:
    print(x, end=" ")
    x -= 1
else:
    print("\nDone!")
```

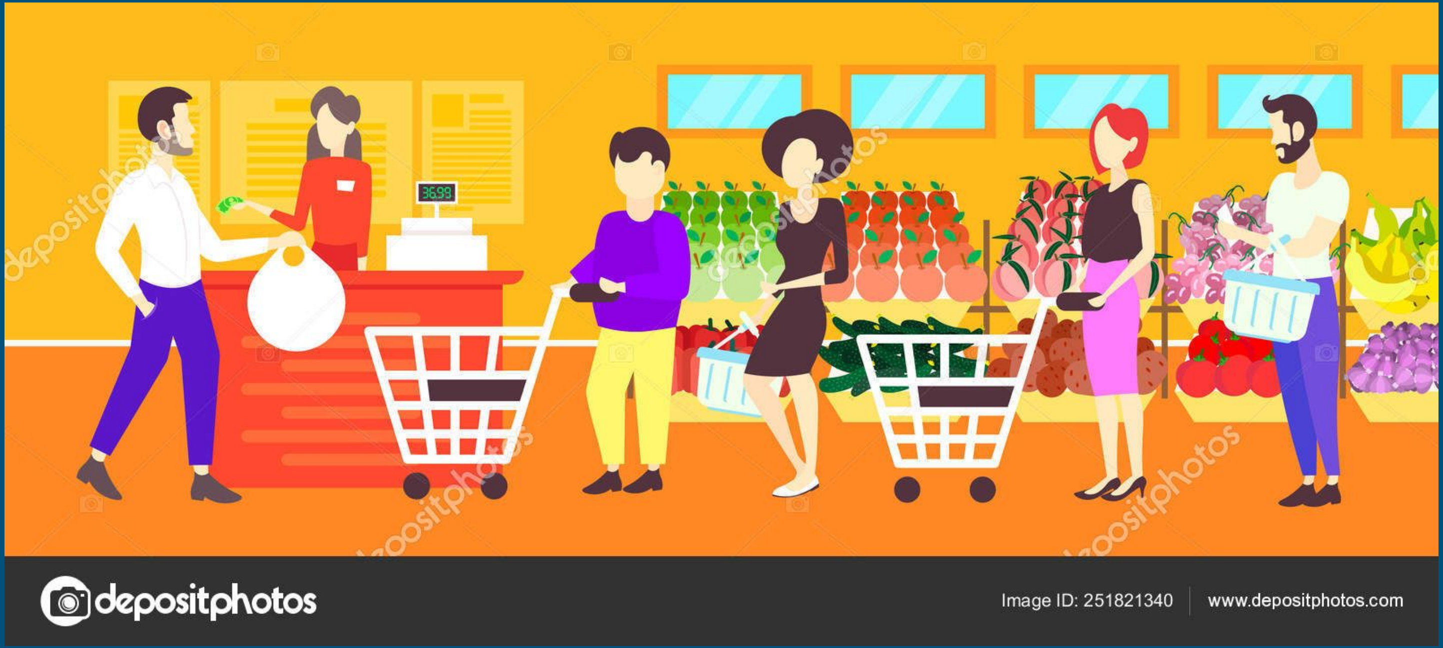
```
6 5 4 3 2 1
Done!
```

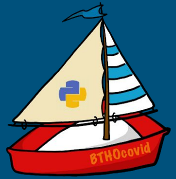


```
# Print all even integers smaller than 10
x = 0
while x < 10:
    print(x, end=" ")
    x += 2
print("\nDone and else was optional!")
```

```
0 2 4 6 8
Done and else was optional!
```

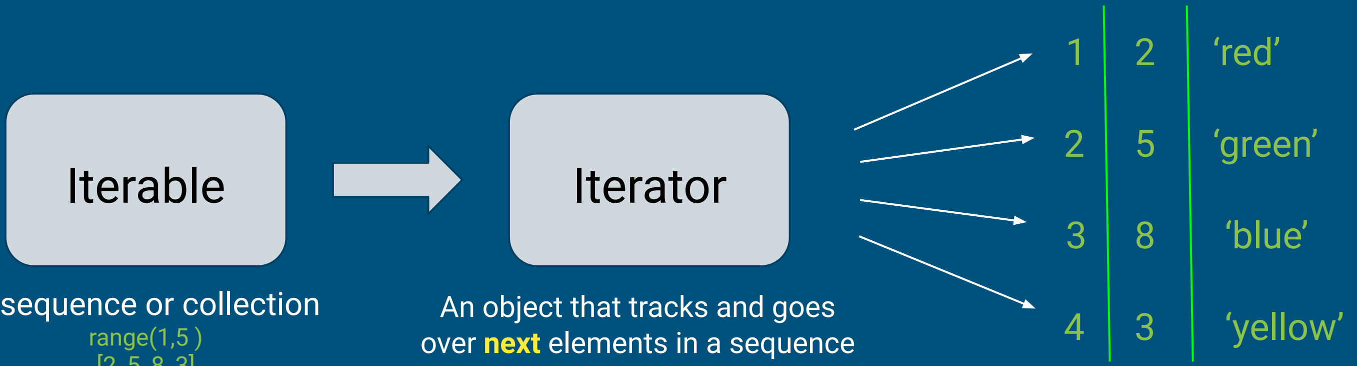
Iterating through a sequence





Iterables and Iterators

An **iterable** is any Python object capable of returning its elements/members one at a time, permitting it to be iterated over using an **iterator**.



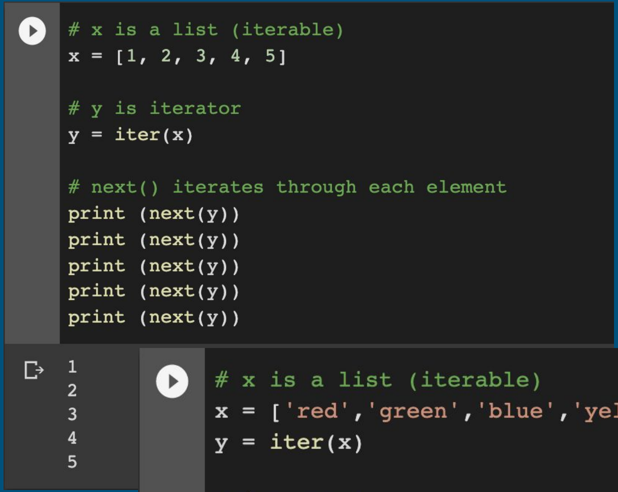
A sequence or collection
range(1,5)
[2, 5, 8, 3]
['red', 'green', 'blue', 'yellow']

An object that tracks and goes over **next** elements in a sequence or collection one at a time

Value of iterator in every iteration/round while going over the sequence/ collection

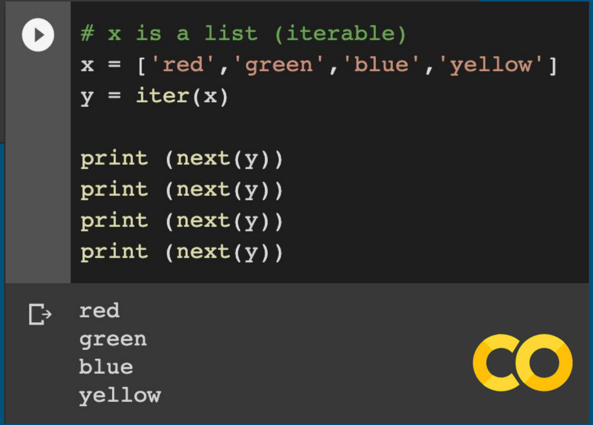
range(start, stop, step) function returns all integers from **start** (default is 0) upto **stop** (not including) with incrementation of **step** (default is 1, can be negative too)

Iterables and Iterators: Examples



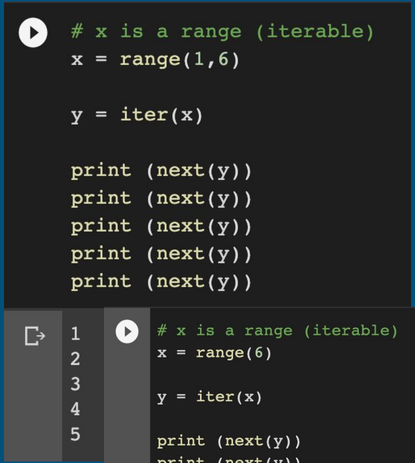
```
# x is a list (iterable)
x = [1, 2, 3, 4, 5]

# y is iterator
y = iter(x)

# next() iterates through each element
print (next(y))
print (next(y))
print (next(y))
print (next(y))
print (next(y))
```

```
1
2
3
4
5
```



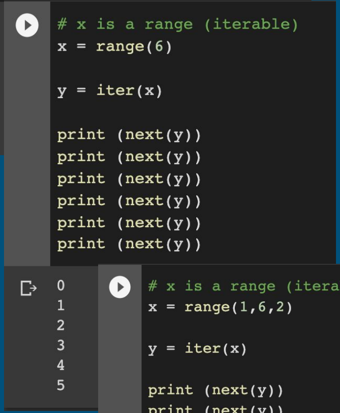
```
# x is a list (iterable)
x = ['red','green','blue','yellow']
y = iter(x)

print (next(y))
print (next(y))
print (next(y))
print (next(y))
```

```
red
green
blue
yellow
```



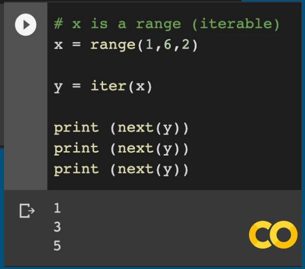
```
# x is a range (iterable)
x = range(1,6)

y = iter(x)

print (next(y))
print (next(y))
print (next(y))
print (next(y))
print (next(y))
```

```
1
2
3
4
5
```



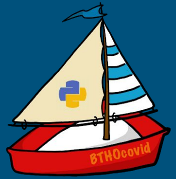
```
# x is a range (iterable)
x = range(6)

y = iter(x)

print (next(y))
print (next(y))
print (next(y))
print (next(y))
print (next(y))
print (next(y))
```

```
0
1
2
3
4
5
```

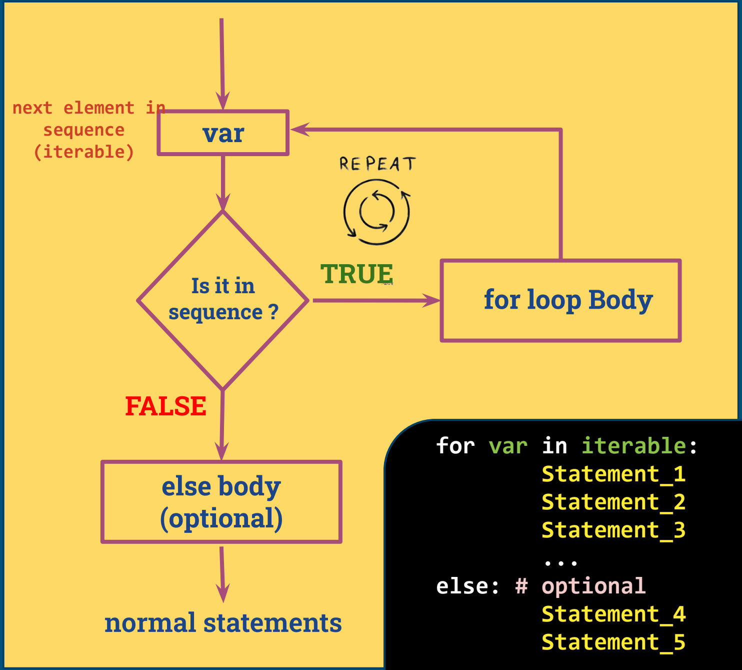


```
# x is a range (iterable)
x = range(1,6,2)

y = iter(x)

print (next(y))
print (next(y))
print (next(y))
```

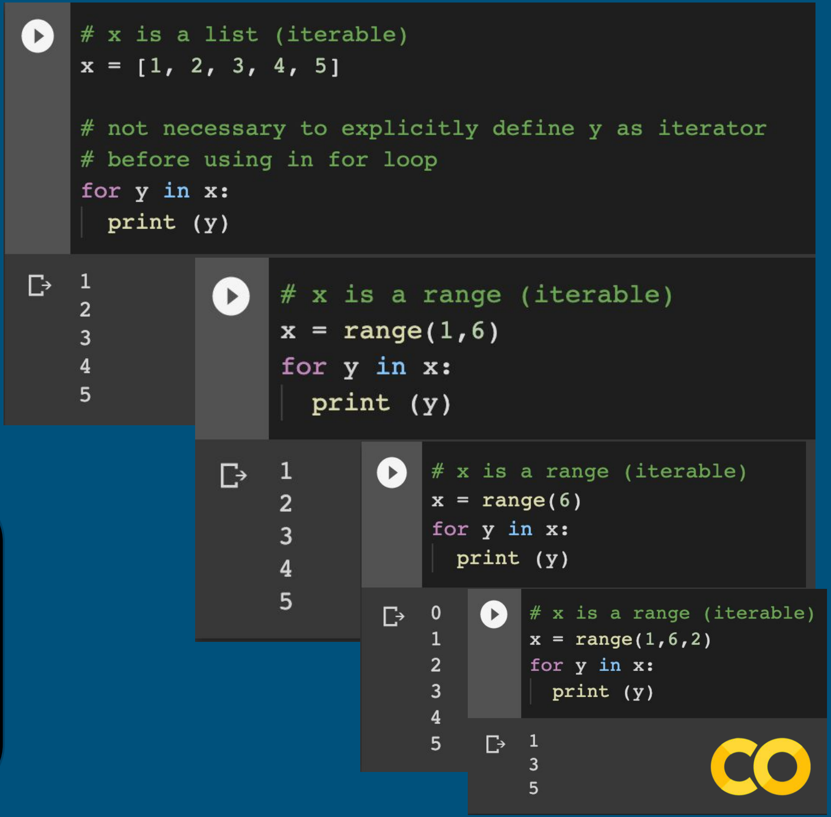
```
1
3
5
```



for loop



```
for var in iterable:
        Statement_1
        Statement_2
        Statement_3
        ...
else: # optional
        Statement_4
        Statement_5
        ...
# normal statements follow
```



```
# x is a list (iterable)
x = [1, 2, 3, 4, 5]

# not necessary to explicitly define y as iterator
# before using in for loop
for y in x:
    print (y)
```

```
1
2
3
4
5
```

```
# x is a range (iterable)
x = range(1,6)
for y in x:
    print (y)
```

```
1
2
3
4
5
```

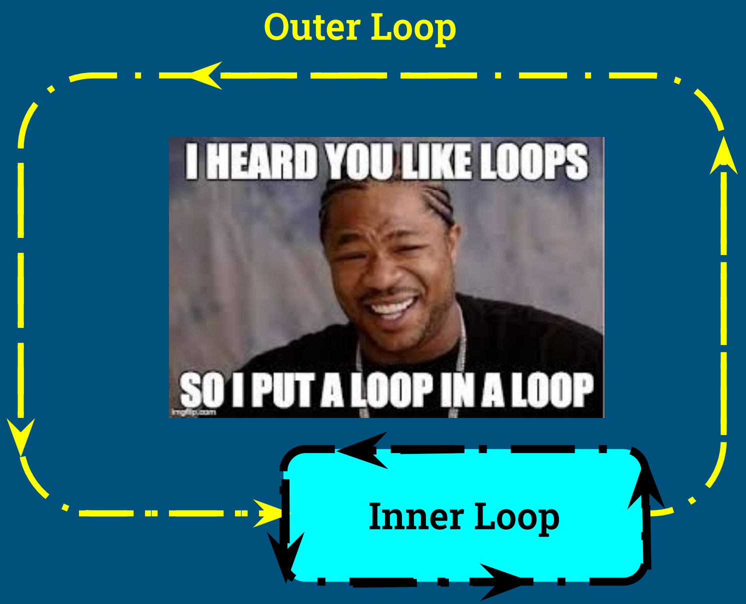
```
# x is a range (iterable)
x = range(6)
for y in x:
    print (y)
```

```
0
1
2
3
4
5
```

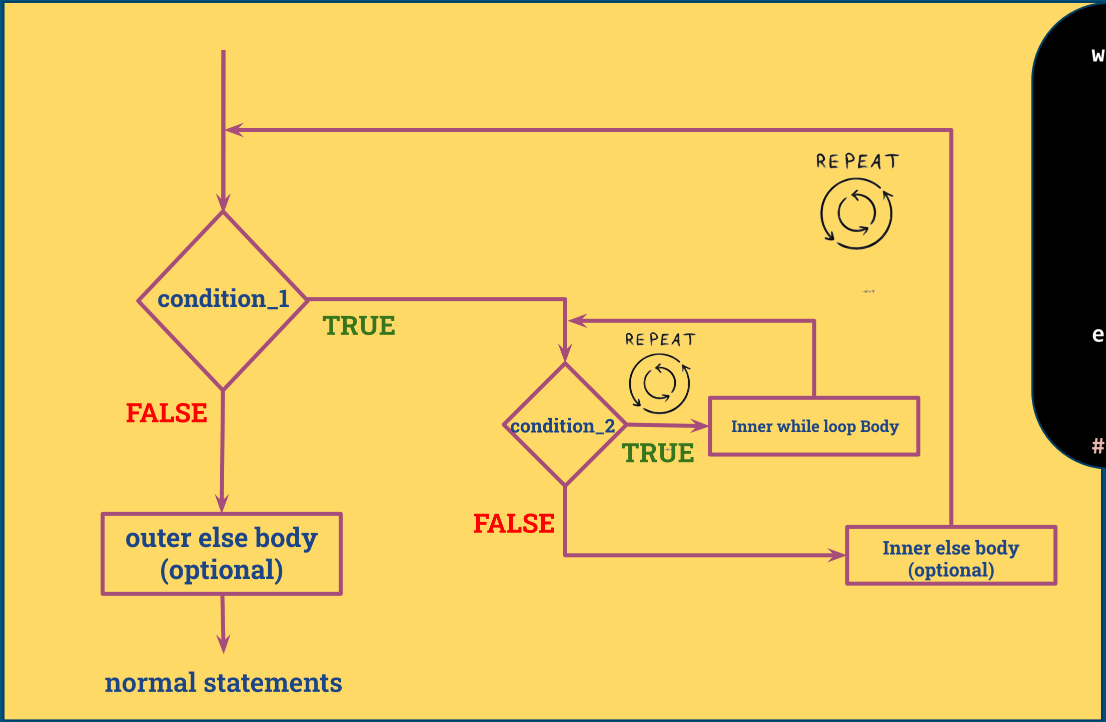
```
# x is a range (iterable)
x = range(1,6,2)
for y in x:
    print (y)
```

```
1
3
5
```

Nested Loops

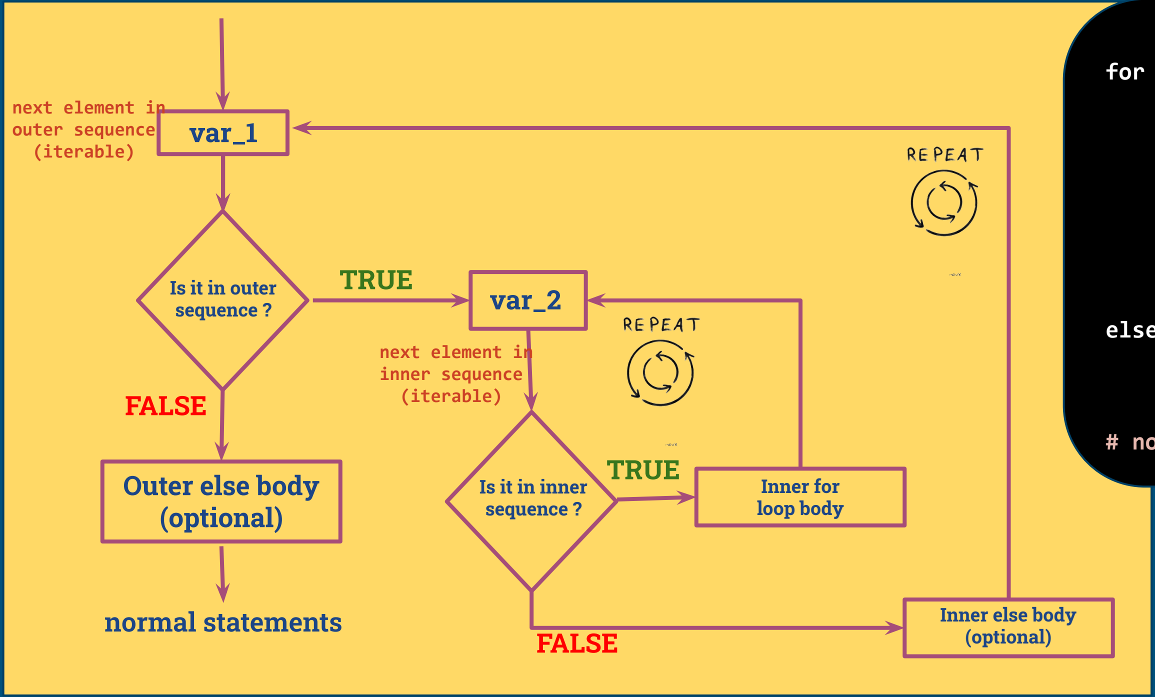


Nested while Loop



```
while condition_1:
        while condition_2:
               inner_while_Statement_1
               inner_while_Statement_2
               …
        else: # inner else optional
               inner_else_Statement_1
               inner_else_Statement_2
               …

else: # outer else optional
       outer_else_Statement_1
       outer_else_Statement_2
       …
# normal statements follow
```

Nested for Loop


next element in outer sequence (iterable)
var_1
Is it in outer sequence ?
TRUE
FALSE
Outer else body (optional)
normal statements
var_2
next element in inner sequence (iterable)
REPEAT
Is it in inner sequence ?
TRUE
Inner for loop body
FALSE
Inner else body (optional)
REPEAT

```
for var_1 in outer_iterable:
        for var_2 in inner_iterable:
                inner_for_Statement_1
                inner_for_Statement_2
                …
        else: # inner else optional
                inner_else_Statement_1
                inner_else_Statement_2
                …
else: # outer else optional
           outer_else_Statement_1
           outer_else_Statement_2
           …
# normal statements follow
```

Altering loop flow - break and continue

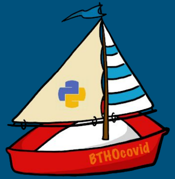
Sometimes we wish to skip the current loop iteration (continue) or terminate the entire loop (break) without continuously checking LOOP condition

break

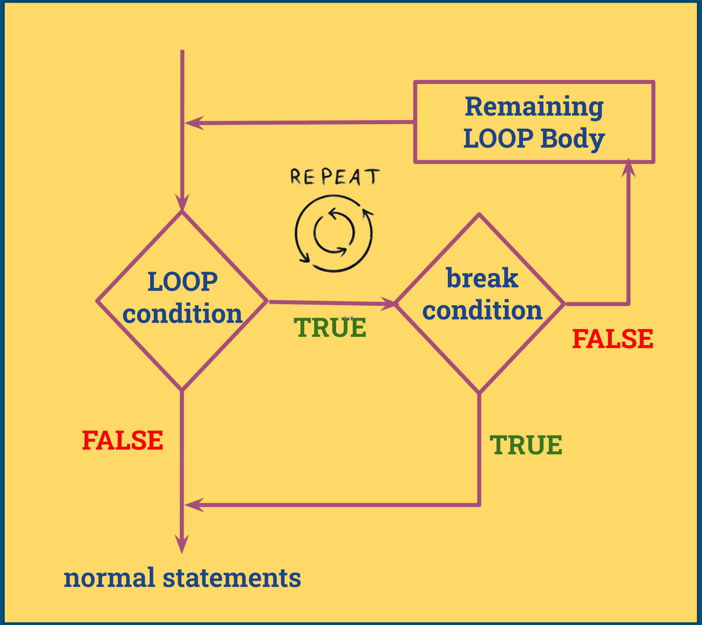
- Terminates the loop containing it and control of the program flows to the statement immediately after the body of the loop.
- If break statement is inside a nested loop, it will terminate the innermost loop
- Saves time by avoiding unnecessary loop iterations when desired outcome is already reached

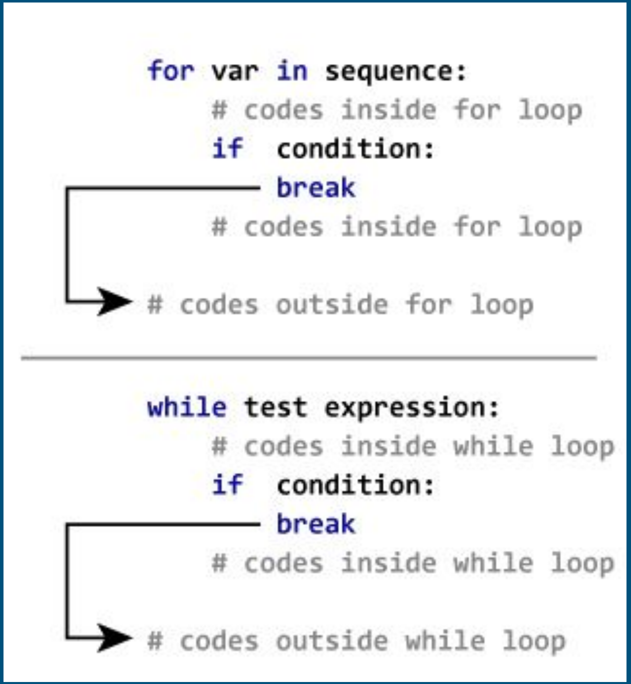
continue

- Skips the rest of the loop body for the current iteration
- Continues on with the next iteration
- Used when you want to skip any iteration and move to next iteration in search of desired outcome



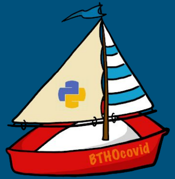
break



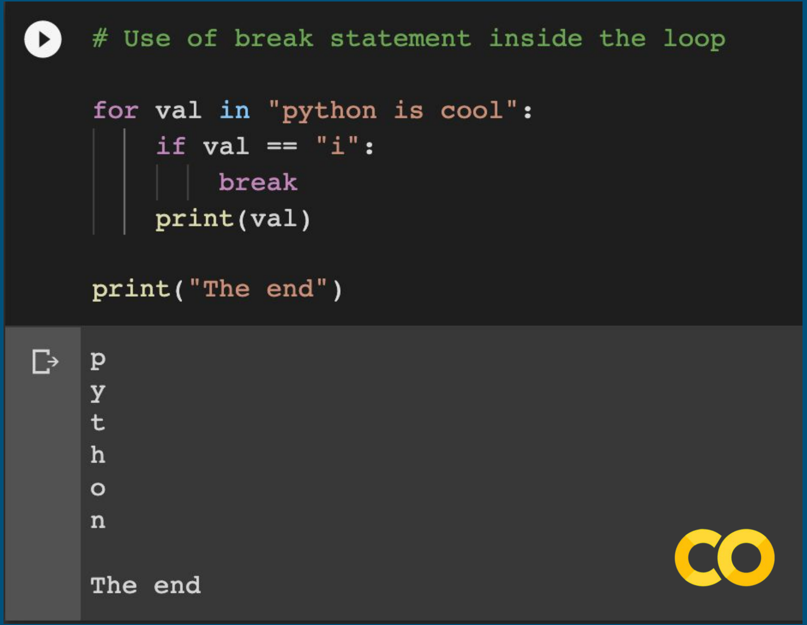


```
for var in sequence:
    # codes inside for loop
    if  condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        break
    # codes inside while loop

# codes outside while loop
```

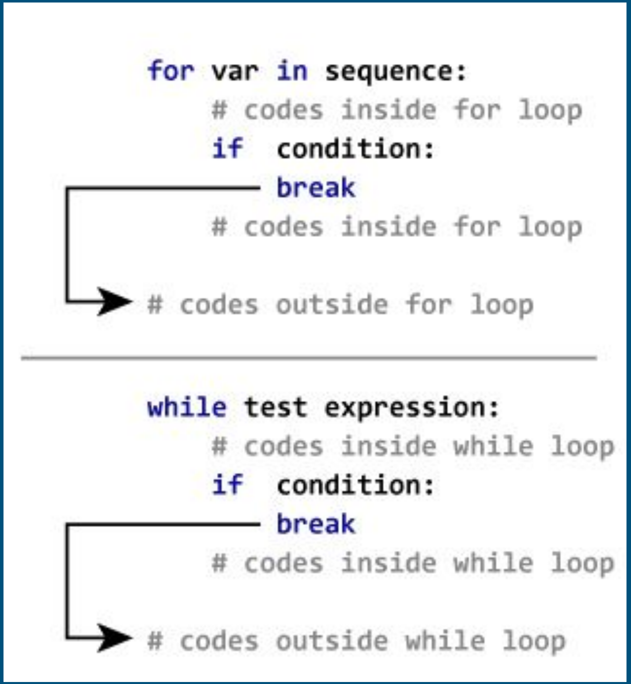


break: Example



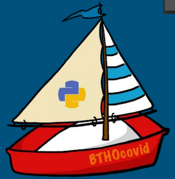
```
# Use of break statement inside the loop

for val in "python is cool":
    if val == "i":
        break
    print(val)

print("The end")
```

```
p
y
t
h
o
n

The end
```

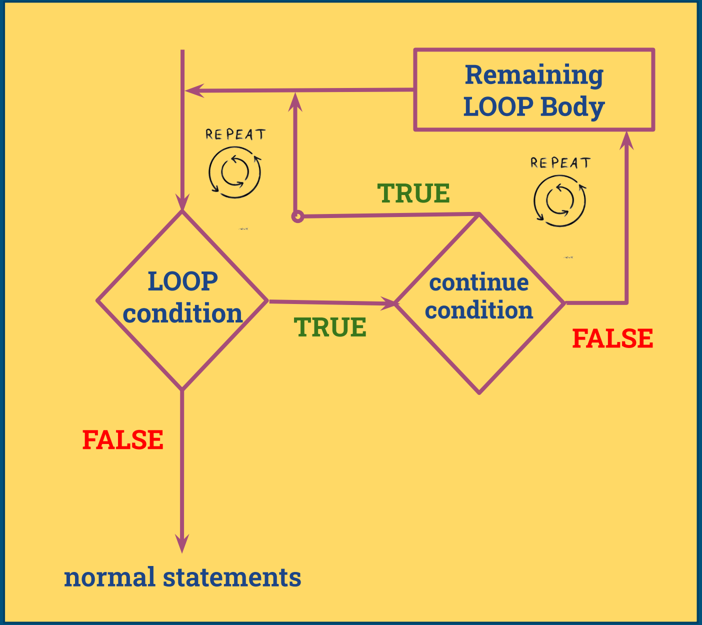


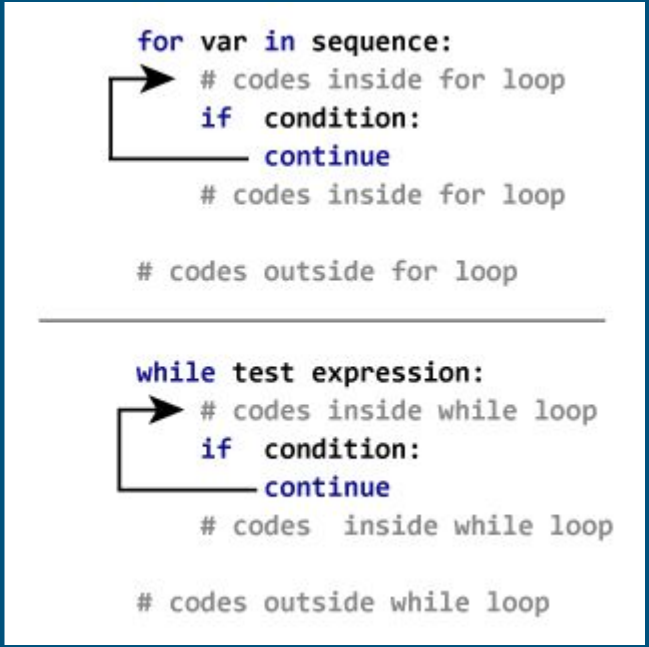
```
for var in sequence:
    # codes inside for loop
    if condition:
        break
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if condition:
        break
    # codes inside while loop

# codes outside while loop
```



continue

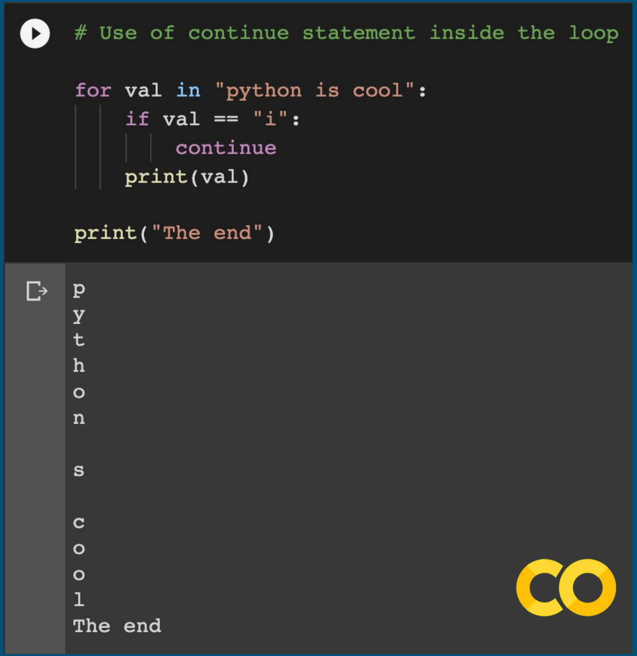




```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

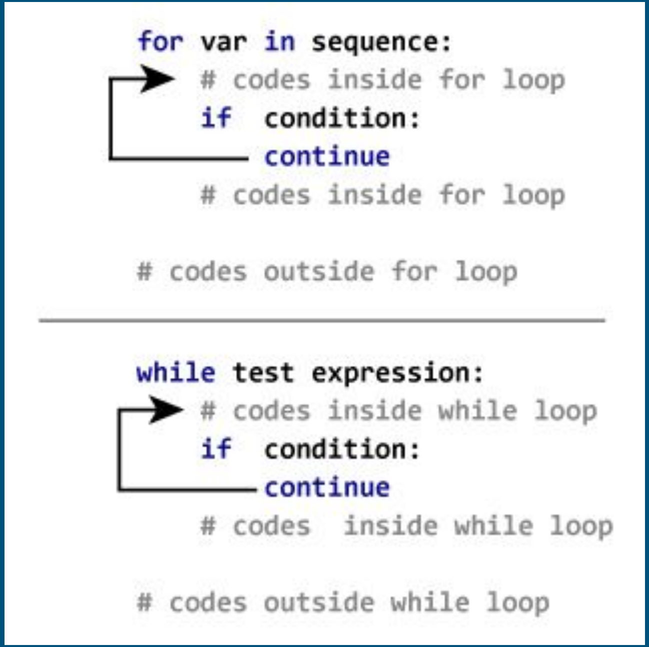
```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```

continue: Example



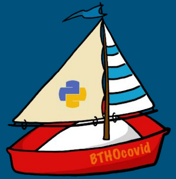
```
# Use of continue statement inside the loop

for val in "python is cool":
    if val == "i":
        continue
    print(val)

print("The end")
```

```
p
y
t
h
o
n

s

c
o
o
l
The end
```



```
for var in sequence:
    # codes inside for loop
    if  condition:
        continue
    # codes inside for loop

# codes outside for loop
```

```
while test expression:
    # codes inside while loop
    if  condition:
        continue
    # codes  inside while loop

# codes outside while loop
```



Next Lecture

Collective Data Structures

Wed, April 29 (9 pm-10 pm CDT)