ITP 115 – Programming in Python

Strings as Sequences

Loops

Outline

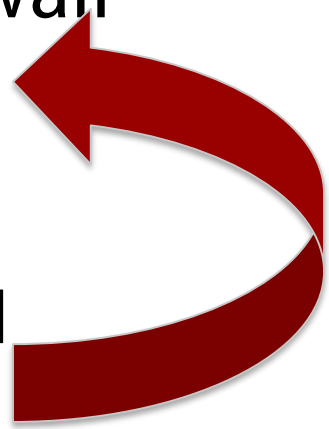
- `while` loops
- `for` loops
- `range()` function
- strings as sequences
- sequence functions and operators

Consider…

- Why might we want code to repeat?

Loop Motivation #1: We want to do things multiple times

- 99 bottles of cold brew coffee on the wall
  - Take one down
  - Pass it around
  - 98 bottles of cold brew coffee on the wall



Do again until you have 0 bottles on the wall

Pseudocode

```
bottles = 99

while
```

Pseudocode

```
bottles = 99

while bottles is greater than 0
    sing bottles on wall
    take one down (bottles - 1)
    pass it around
    repeat again
```

Loop Motivation #2: Users make mistakes

- It is easy for users to make mistakes when they have to type "answers"
- Ideally, we should ask them to try again

```
Do you want cream (y/n)? Q
Oops! Invalid choice

<program ends>
```

vs

```
Do you want cream (y/n)? Q
Oops! Invalid choice

Do you want cream (y/n)? Y
```

How would you describe the fix?

```
while
```

_____

_____

Pseudocode

```
while
The user has not entered "y" or "n"
Ask the user for a value of "y" or "n"
```

`while` Loops

- Loops let us repeat something
- `while` loop is similar to the `if` structure
- As long as the **condition** is `true`, the block (*loop body*) is executed
- When the condition is `false`, the loop exits and the program continues

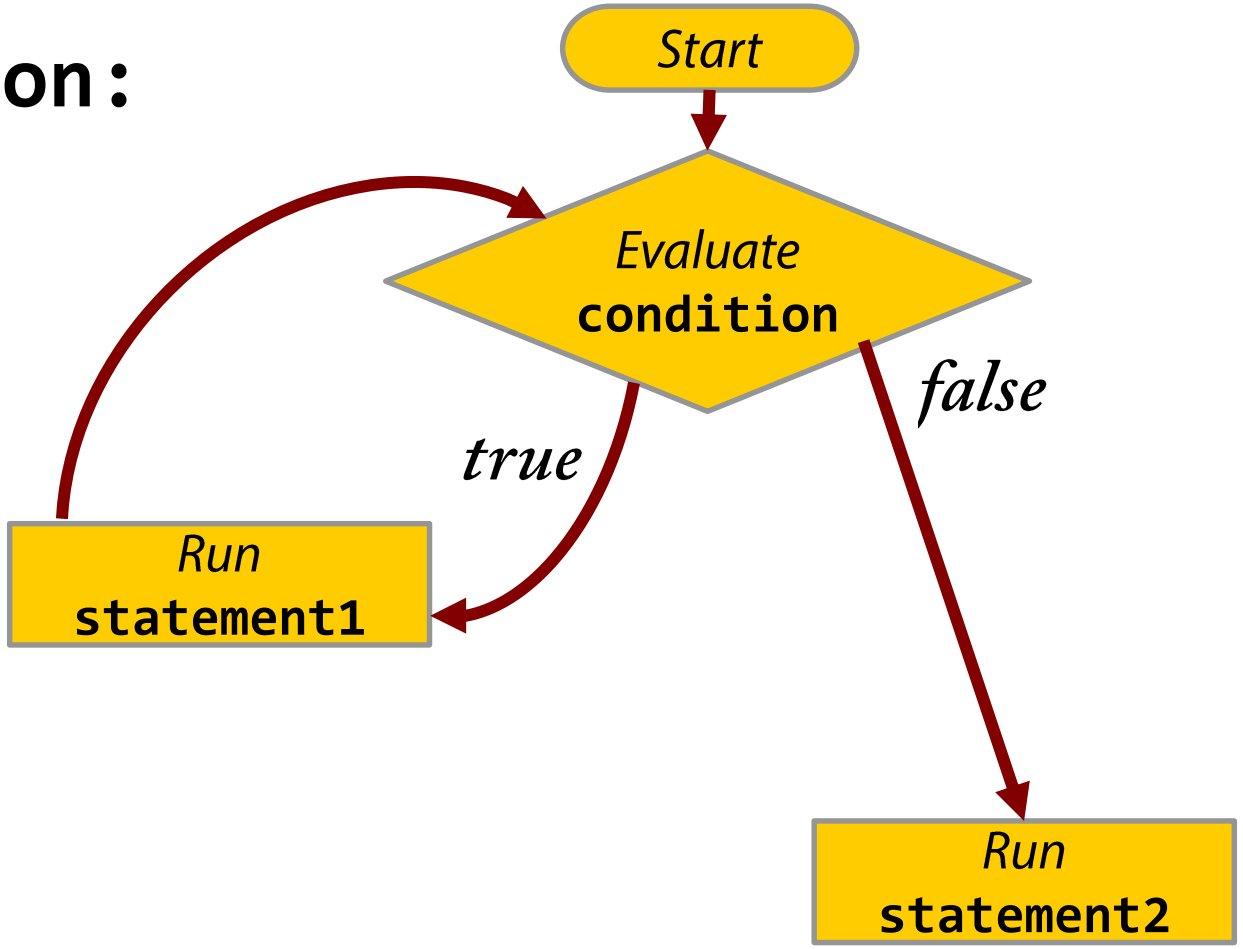
The `while` Statement

- Syntax

```
while condition:
        statement1
        statement2
```

Semantics of `while`

```
while condition:
    statement1

statement2
```



Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

Check this condition at the start of each loop iteration

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

Do this if the condition is **`true`** at the beginning of the block

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

When you reach the end of the **while** block, go to the top again

Motivation #1: Multiple Times

```
numBottles = 99

while numBottles > 0:
    print(numBottles, "bottles of cold brew coffee on the wall")
    print("Take one down and pass it around")
    numBottles = numBottles - 1
    print(numBottles, "bottles of cold brew coffee on the wall")

print("Cold brew coffee is all gone!")
```

Check this condition at the again

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
      answer = input("Do you want cream (y/n): ")


# Otherwise go on with the calculations
```

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
    answer = input("Do you want cream (y/n): ")


# Otherwise go on with the calculations
```

Check this condition at the start of each loop iteration

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
      answer = input("Do you want cream (y/n): ")

# Otherwise go on with the calculations
```

Do this if the condition is **true** at the beginning of the block

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
      answer = input("Do you want cream (y/n): ")

 ----------------------------------------

# Otherwise go on with the calculations
```

When you reach the end of the **while** block, go to the top again

Motivation #2: User Input Error

```
answer = input("Do you want cream (y/n): ")

while answer.lower() != "y" and answer.lower() != "n":
    answer = input("Do you want cream (y/n): ")


# Otherwise go on with the calculations
```

Check this condition at the start of next iteration (loop)

Consider…

```
counter = 10
while counter > 0:
    print(counter)
    counter = counter - 1
print("Blast OFF!")
```

Consider…

```
counter = 10
while counter > 0:
    print(counter)
    counter = counter + 1
print("Never!")
```

Infinite Loops

- A loop that always repeats without ending is an **infinite loop**
- This happens when the boolean condition never becomes false
- There should always be a way to exit a loop

Infinite Loops Examples

```
counter = 10
while counter > 0:
      print(counter)
      counter = counter + 1
```

```
word = input("Enter a word: ")
while word != "exit":
      print(word)
```

```
while True:
        print("Wheee!")
```

Reference only

`break` and `continue` Statements

- The **`break`** statement means "*break out of the loop*"
- The **`continue`** statement means "*jump back to the top of the loop*"
- We will **not** use break and continue
  - Though sometimes useful, they can lead to poor understanding of loops

Reference only

Example

```
count = 0
while True:
    count += 1

    # end loop if count >= 10
    if count > 10:
        break

    # skip 5
    if count == 5:
        continue

    print(count)
```

```
1
2
3
4
6
7
8
9
10
```

Tips for Designing a Loop Body

- Write out the algorithm in pseudocode
- Look for a repeated pattern—this is **loop body**
  - May not be first action
- Look for variables to initialize **BEFORE** loop
- Look for things to do **AFTER** the loop ends

Tips: Initializing Statements

- Some values are set **before** the loop begins
  - Often these variables get a value of **zero** or **one**
  - Other times it's based on 1 iteration through the loop

- Other variables get values only when the loop is executing

Tips: Updating Variables

- A while loop is controlled by a Boolean condition (often related to a variable)
  - "As long as" that condition is True, the loop continues

- Inside the loop body, you <u>must</u> do something that gets you closer to ending the loop
  - Otherwise we have an infinite loop

Common Loops: Count-Controlled

```
numBottles = 99

while numBottles > 0:
  print(numBottles, "bottles of coffee on the wall")
  numBottles = numBottles - 1
```

Initialization

Update

Common Loops: Sentinel Value

- Sentinel value can be used mark the end of a list
  - Should be different from all other input

- Common sentinel values

  `-1` after a long list of positive numbers

  `1 5 100 0 -1`

  `exit` after other commands

  `show get delete exit`

Common Loops: Sentinel Value

Initialization

```
nextNum = int(input("Enter a num or -1 to quit"))
while nextNum != -1:
      print("num entered = " + nextNum)
      nextNum = int(input("Enter a num or -1 to quit"))

print("You entered -1.")
```

Update

Common Loops: Boolean Value

- The loop condition is a Boolean expression
- Within the loop body there is some action which affects the Boolean expression

Common Loops: Boolean Value

Initialization

```
sum = 0
areMore = True
print("Enter positive numbers or -1 to quit")
while areMore == True:
    nextNum = int(input("Enter a number: "))
    if (nextNum < 0)
        areMore = False
    else
        sum = sum + nextNum
print("The sum is " + sum)
```

Update

```
Enter numbers or -1 to quit
1 2 3 -1
The sum is 6
```

- *End lecture*

`for` Loops

- The `for` loop repeats like `while`, but not based on a condition
- Repeats part of a program based on a **sequence**
- A `for` loop repeats its loop body for each element of the sequence, in order
  - When it reaches the end of the sequence, the loop ends

Creating a `for` Loop

- Start with the word `for`
- Follow it with a new variable *(this variable will only be used for this loop)*
- Follow it with the reserved word `in`
- Follow it with the **sequence** you want to loop through

Creating a `for` Loop

Syntax

```
for variable in sequence:
    # do code in every loop

# do code after the loop
```

Aside: What is a sequence?

- A **sequence** is an "*ordered set of things*"
- Basically, a group of items stored together in a collection
- Examples in Python
  - a "range" of numbers
  - string variables
  - lists *(more on this next week)*
  - tuples *(more on this next week)*

Create Sequences with `range()`

- `range()` function will generate a sequence of numbers based on some parameters

Using `range()`

- **`range(int stop)`**
  - Returns sequence that
    begins at **`0`**
    increases each time by **`1`**
    count up to but not including **`stop`**
  - Ex:

```
range(6) →   0, 1, 2, 3, 4, 5
```

Using `range()`

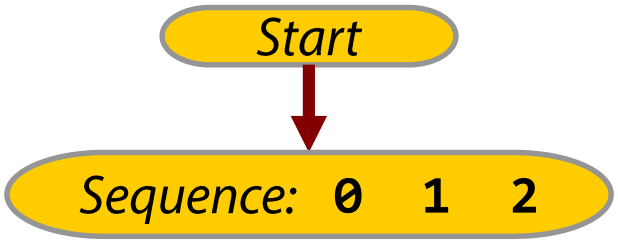
- **`range(int start, int stop, int step)`**
  - Returns sequence
    begins at **`start`**
    increases each time by **`step`**
    counts up to but not including **`stop`**
  - Ex:

```
range(10, 25, 5) →    10, 15, 20
range(10, 26, 5) →    10, 15, 20, 25
```

How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```

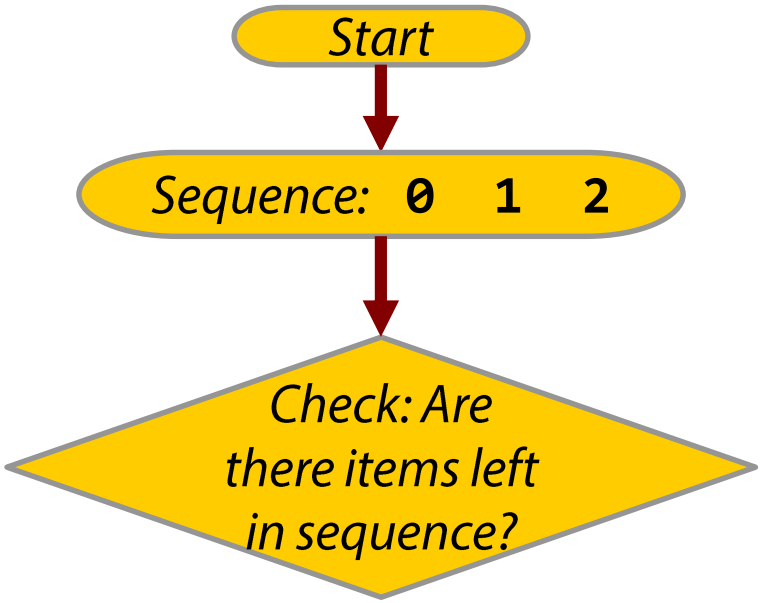
How do `for` loops work?




```
for num in range(3):
    print(num + 2)

print("Done!")
```

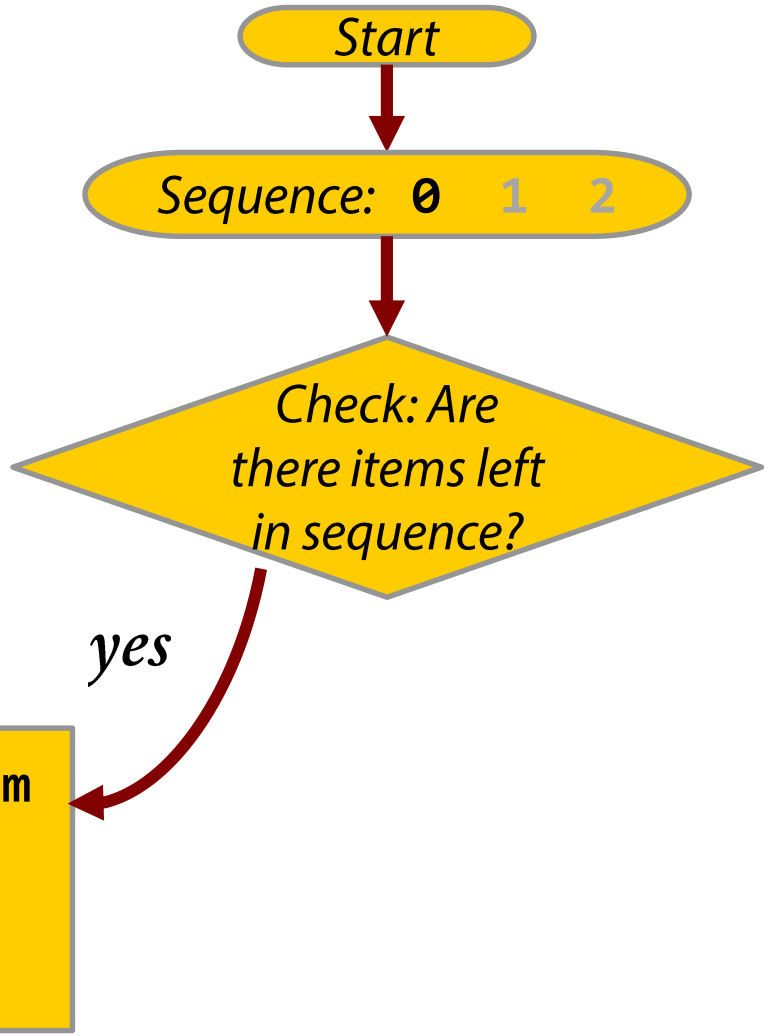
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



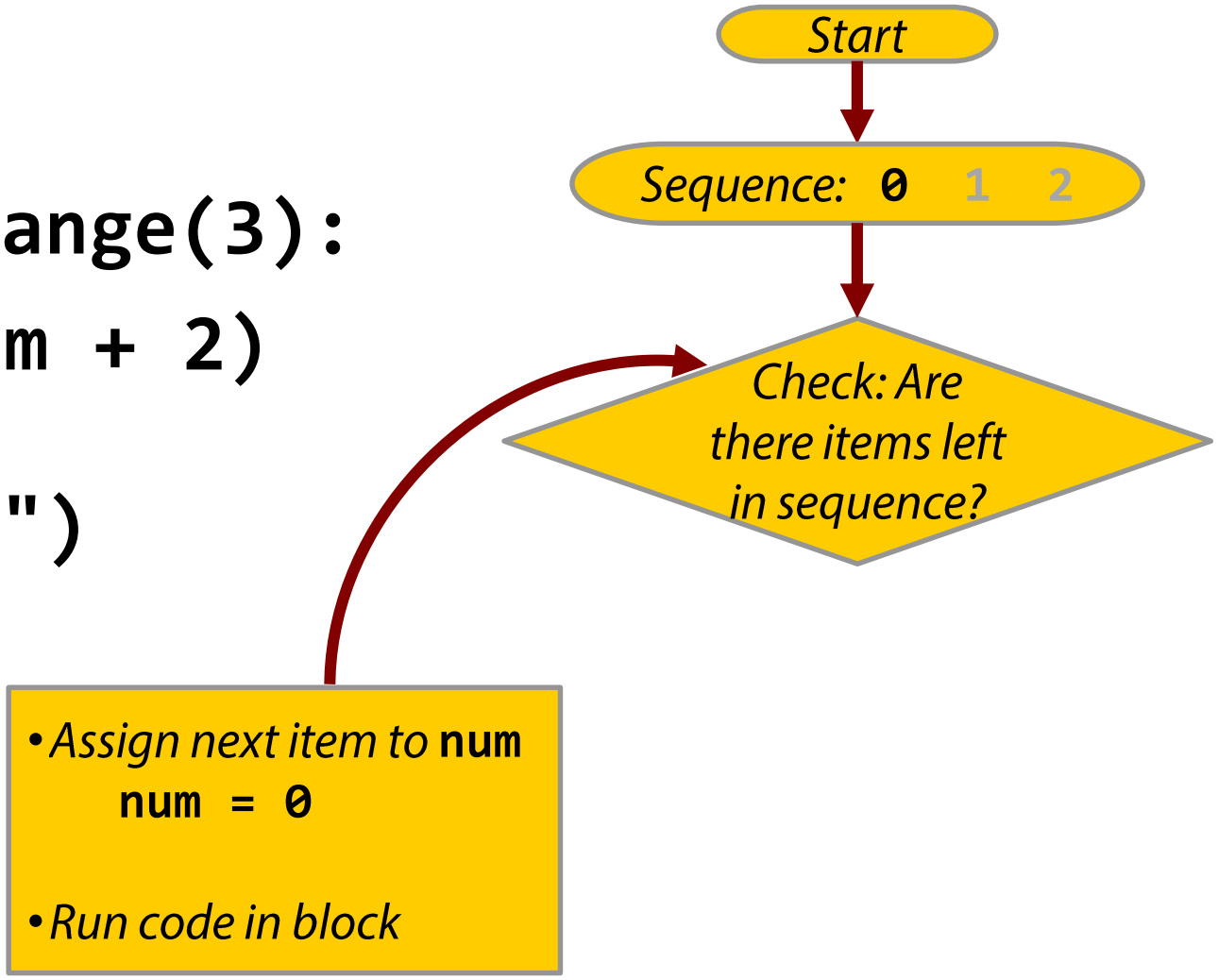
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



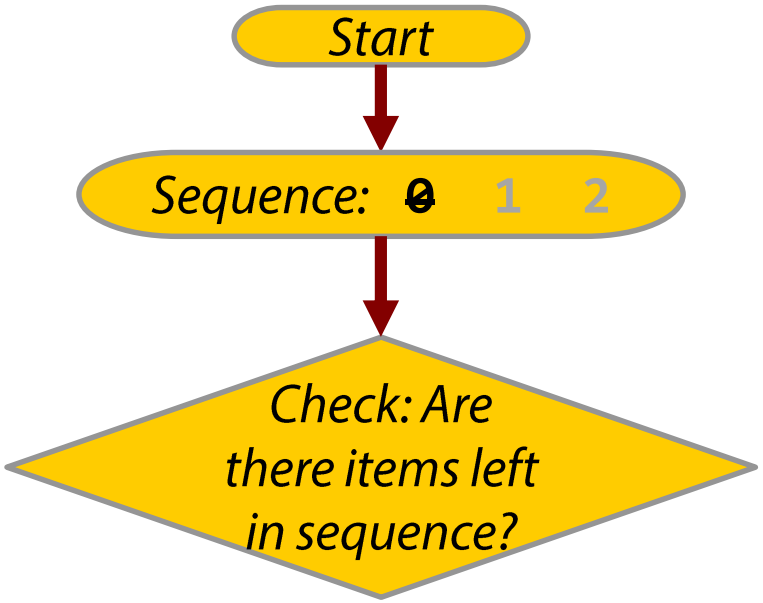
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



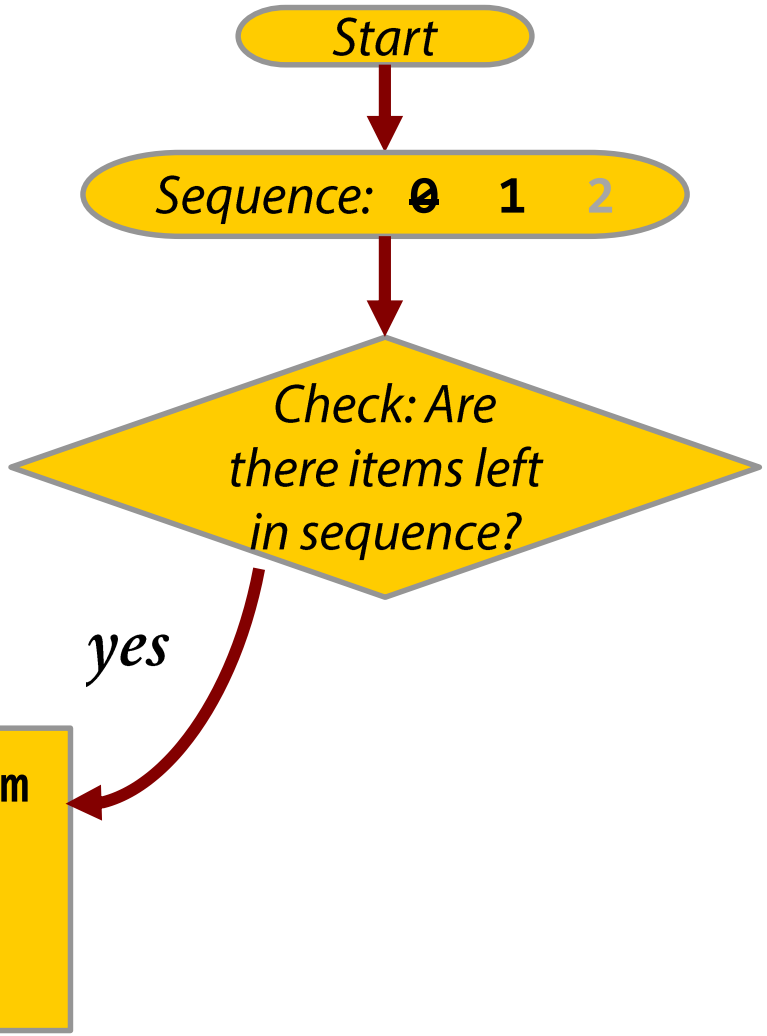
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



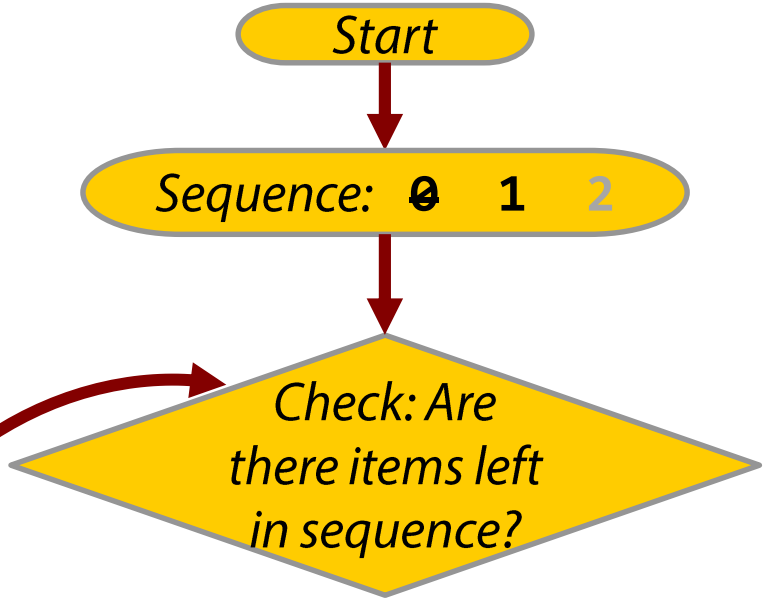
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



How do `for` loops work?

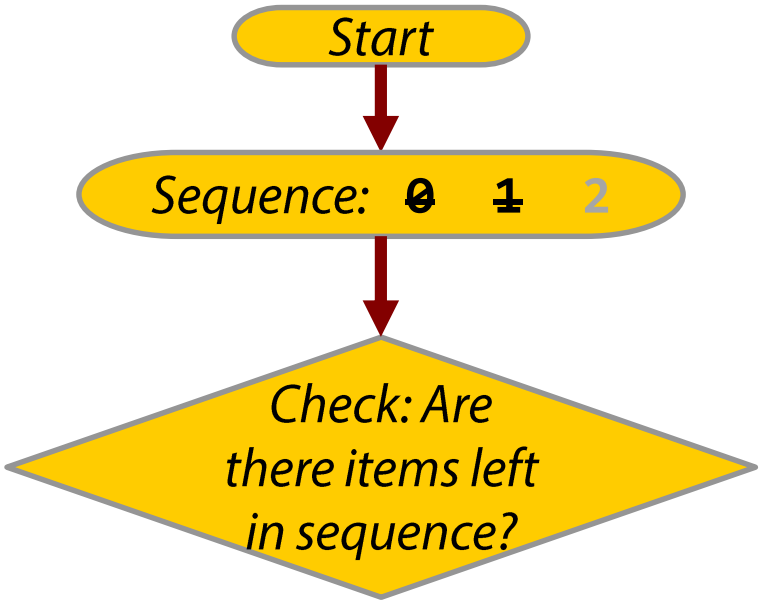
```
for num in range(3):
    print(num + 2)

print("Done!")
```



- *Assign next item to* **num**
  `num = 1`
- *Run code in block*

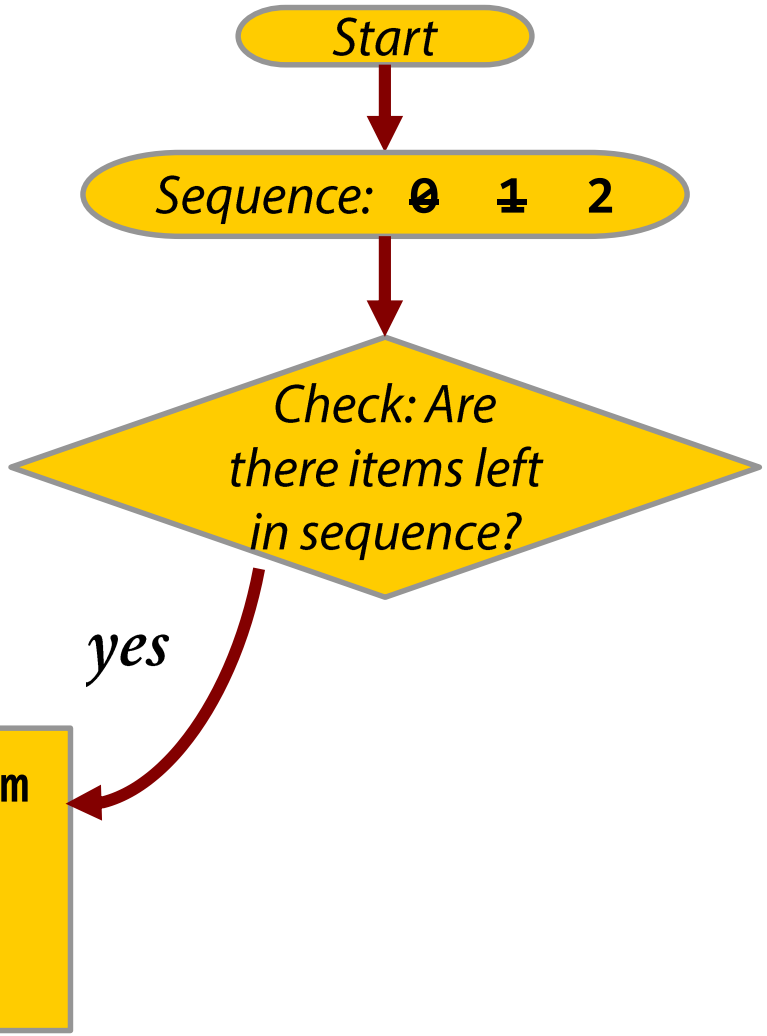
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



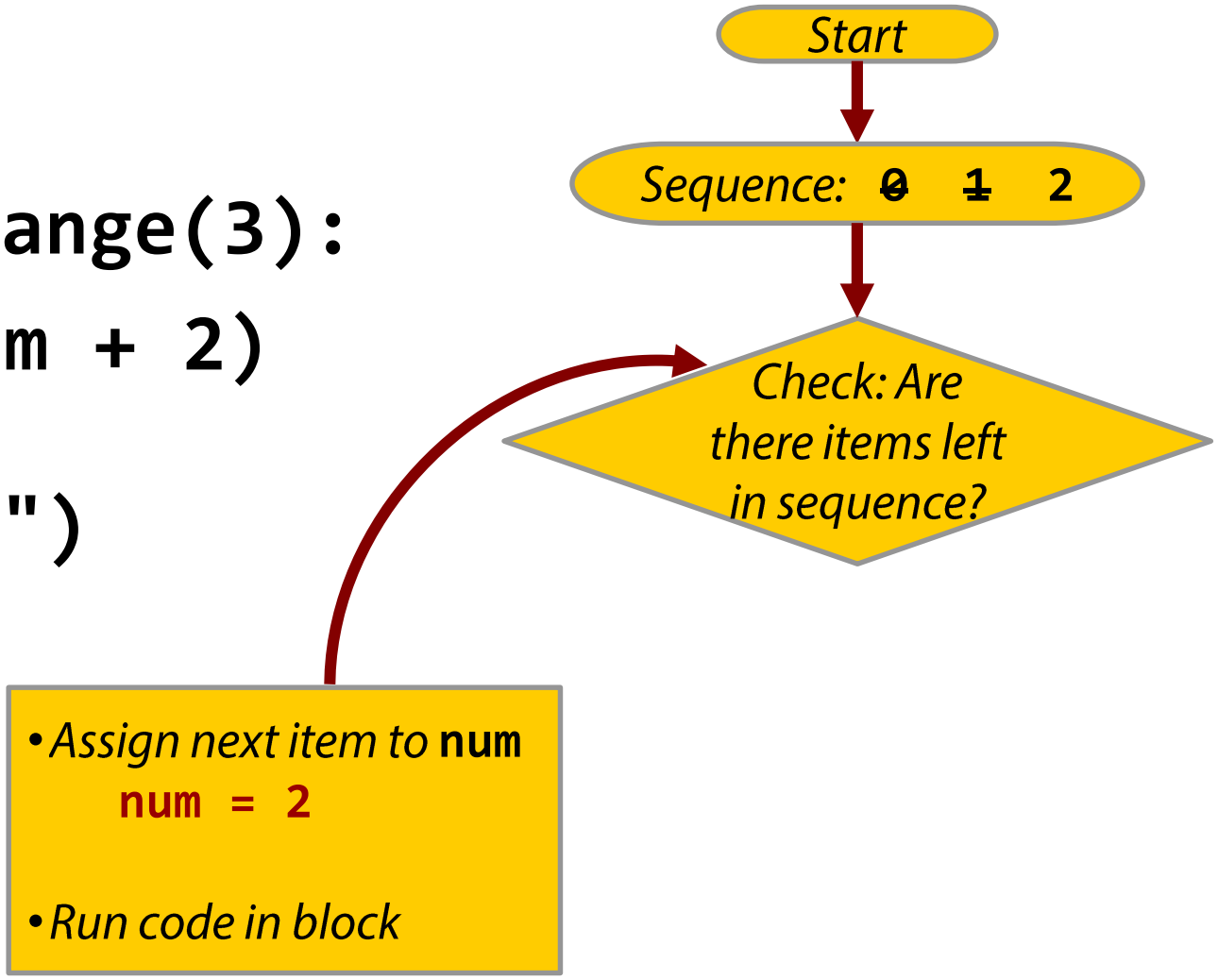
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



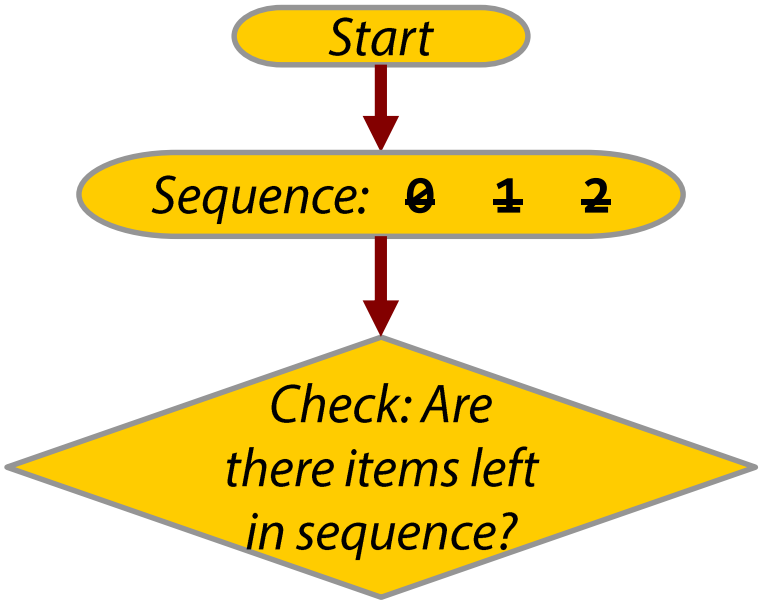
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



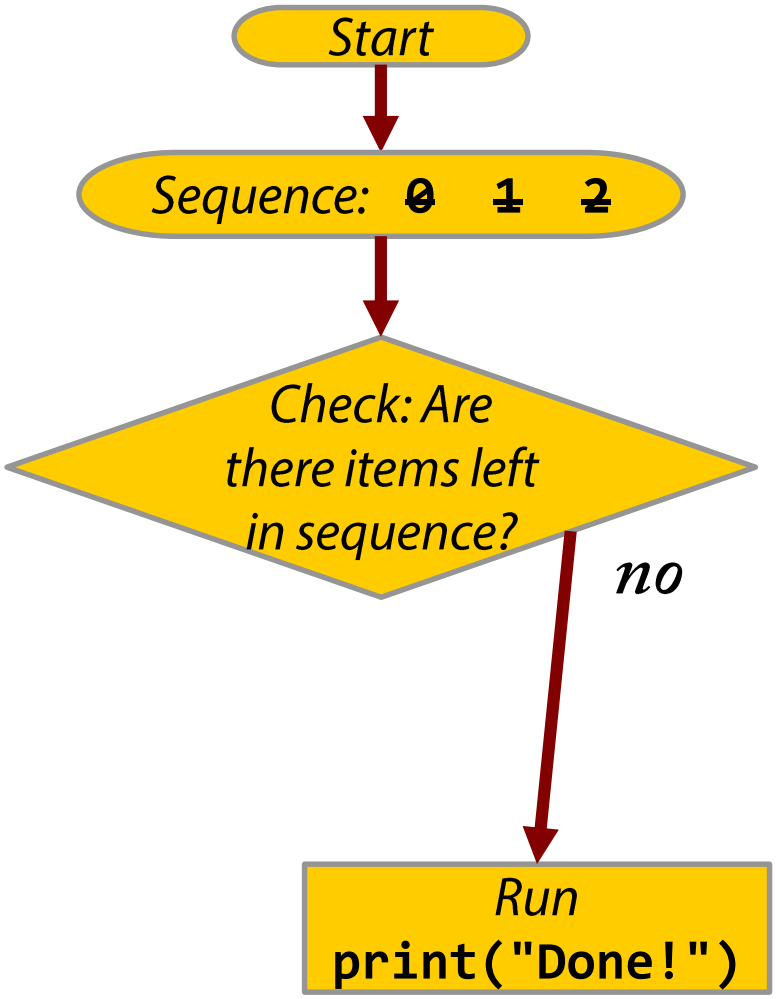
How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



How do `for` loops work?

```
for num in range(3):
    print(num + 2)

print("Done!")
```



Example

```
for i in range(10):
    print(i, end=" ")
```

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

```
for i in range(0, 50, 5):
    print(i, end=" ")
```

```
0 5 10 15 20 25 30 35 40 45
```

```
for i in range(10, 0, -1):
    print(i, end=" ")
```

```
10 9 8 7 6 5 4 3 2 1
```

Strings Are Sequences too!

- We can access strings as entire words (as we have been doing)

```
print("hello")
```

- But since they are sequences, we have two other ways to access them
  - **Sequential access** means going through a sequence one element at a time
  - **Random access** allows you to get any element in a sequence directly (next week)

Strings – Sequential Access

```
msg = "spamalot"

for letter in msg:
    print(letter.upper(), end=" ")
```

Using a string with a **`for`** loop, we can access each letter sequentially *(just as we did with* **`range()`***)*

```
#Output
S P A M A L O T
```

Using `len()` with Strings

`len()` function returns the length of any sequence

Examples:

```
msg = "Hello"
length = len(msg)        # length = 5

msg = "Hello World"
length = len(msg)        # length = 11
```

Using the `in` operator with Strings

`in` operator returns `True` if item is a member of a sequence; `False` if not

```
#Code

msg = "spamalot"
if "spam" in msg:
    print("Found")
else:
    print("NOT Found")
```

```
#Output

Found
```