ITP 115 – Programming in Python

For Loops

Outline

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

`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()`

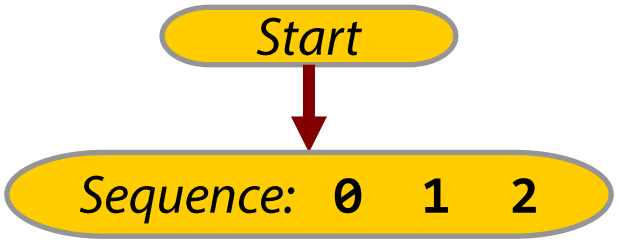
- **`range(int start, int stop, int step)`**
  - Returns sequence
    - begins at **`start`**
    - increases each time by **`step`**
    - counts up to but not including **`stop`**
  - Ex:

```
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!")
```

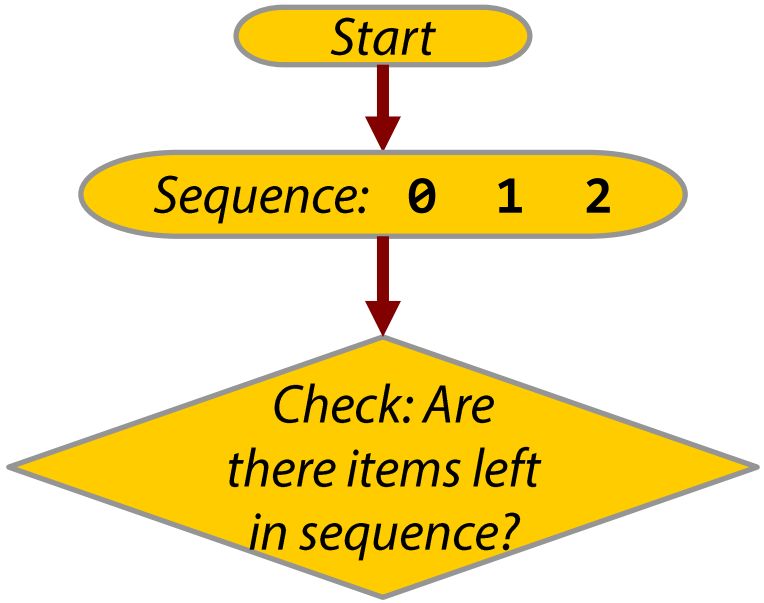
How do `for` loops work?




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

print("Done!")
```

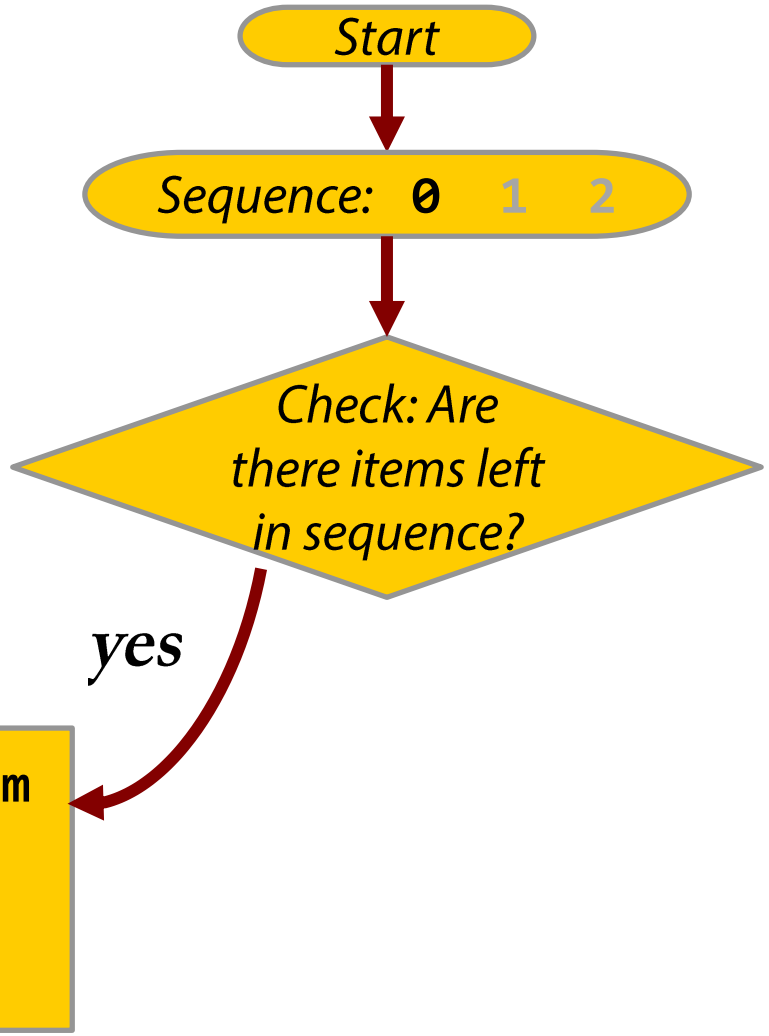
How do `for` loops work?

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

print("Done!")
```



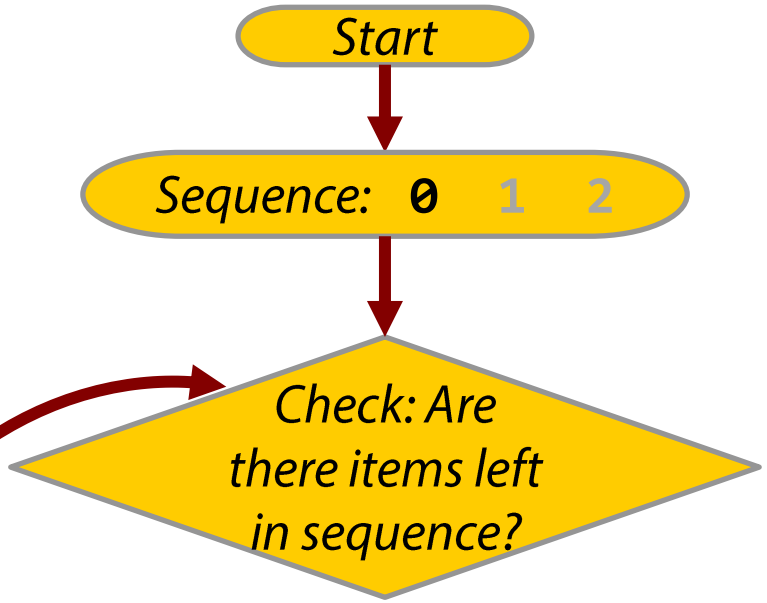
How do `for` loops work?

```
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!")
```



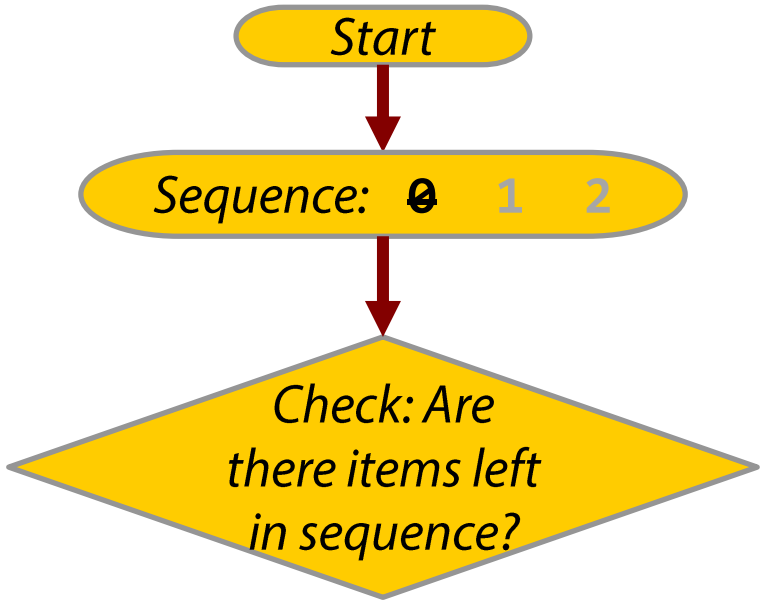
- *Assign next item to* **num**

  `num = 0`

- *Run code in block*

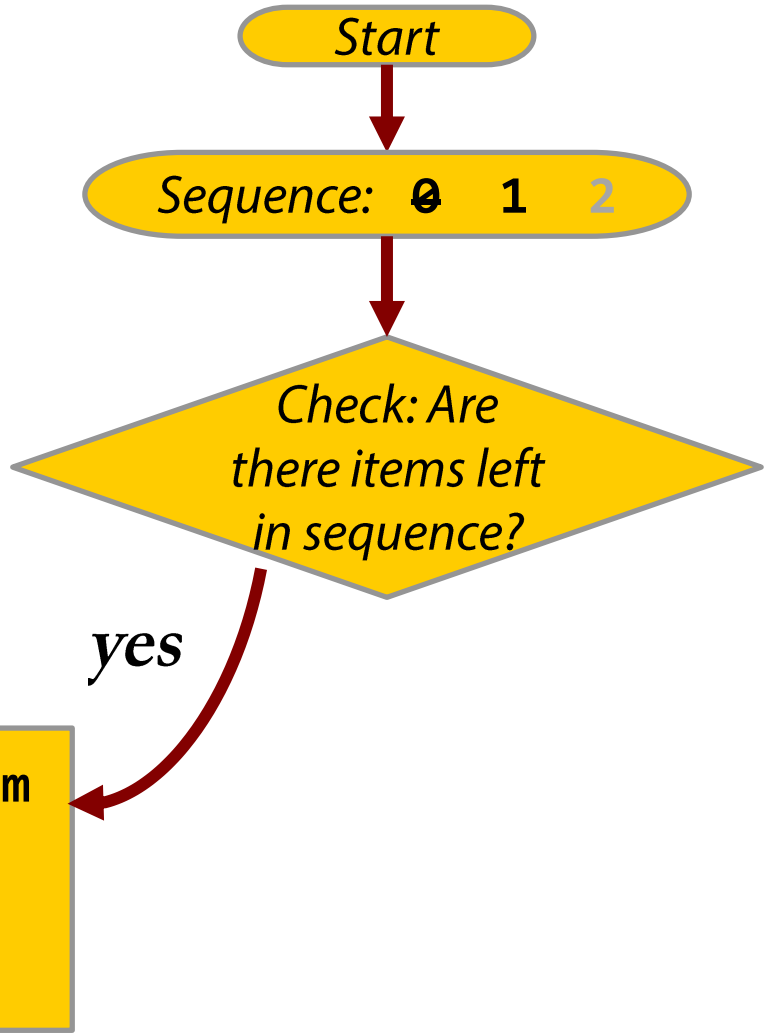
How do `for` loops work?

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

print("Done!")
```



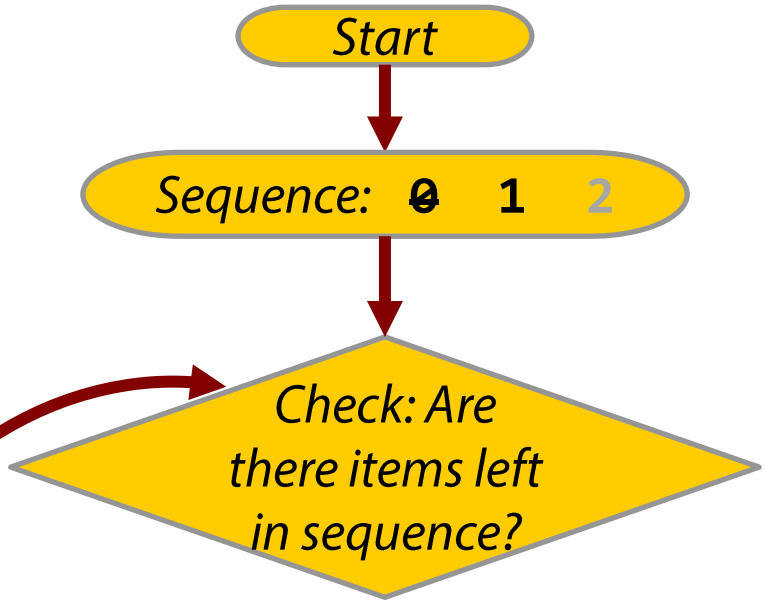
How do `for` loops work?

```
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!")
```



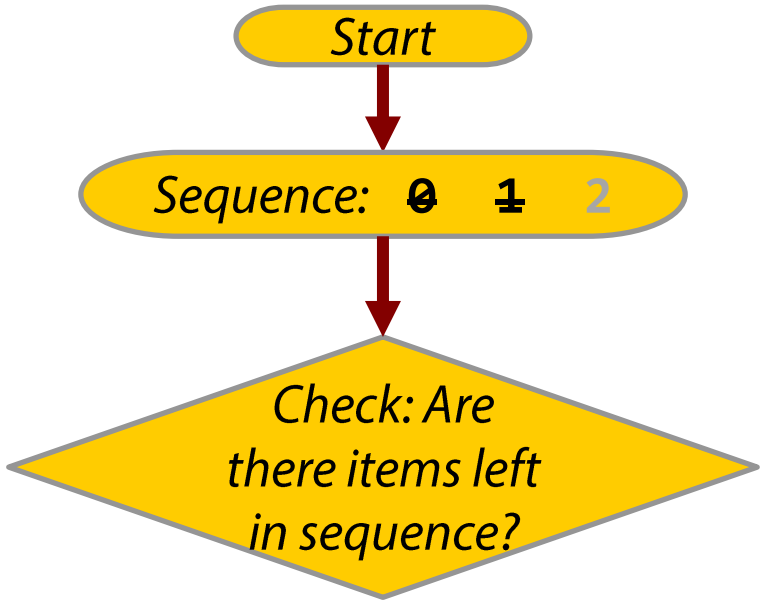
- *Assign next item to* **num**

  `num = 1`

- *Run code in block*

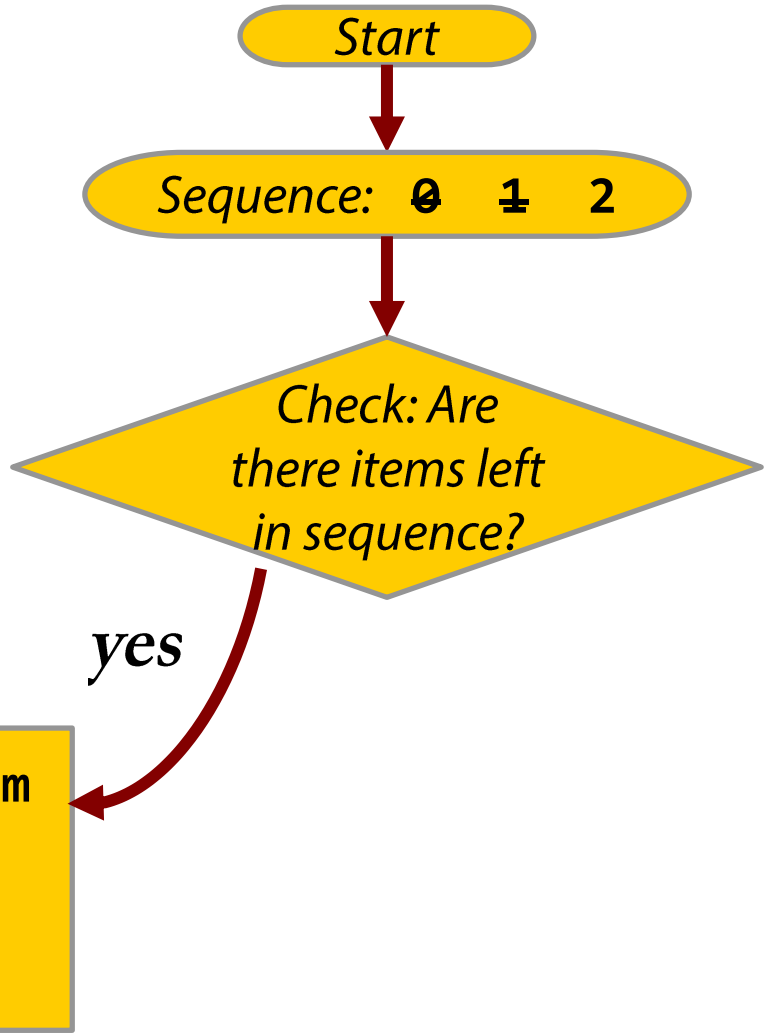
How do `for` loops work?

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

print("Done!")
```



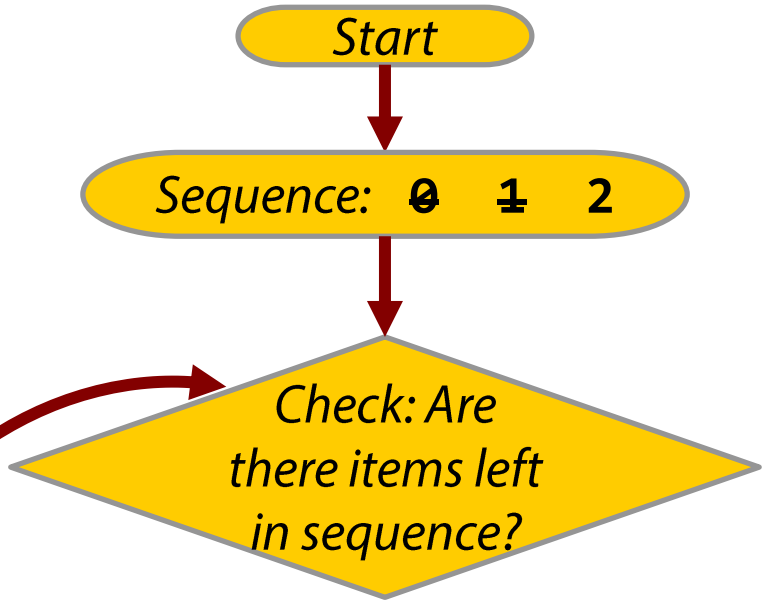
How do `for` loops work?

```
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!")
```



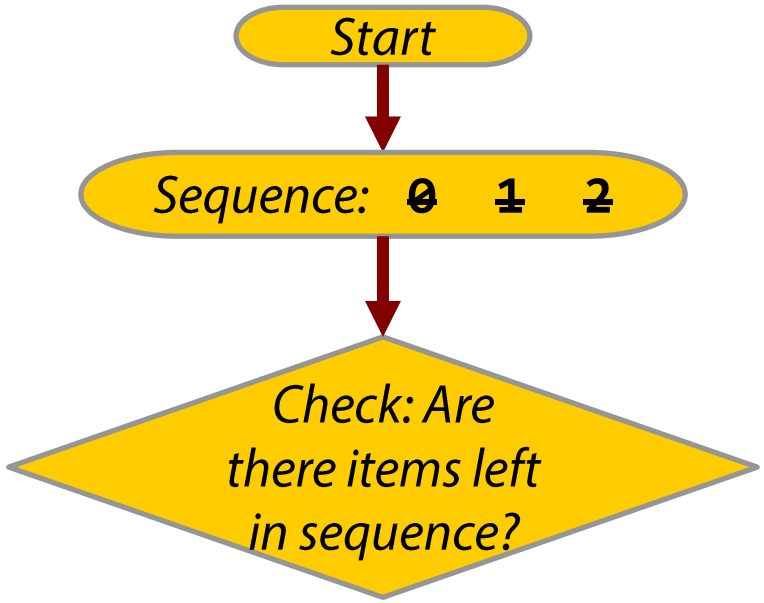
- *Assign next item to* **num**

  `num = 2`

- *Run code in block*

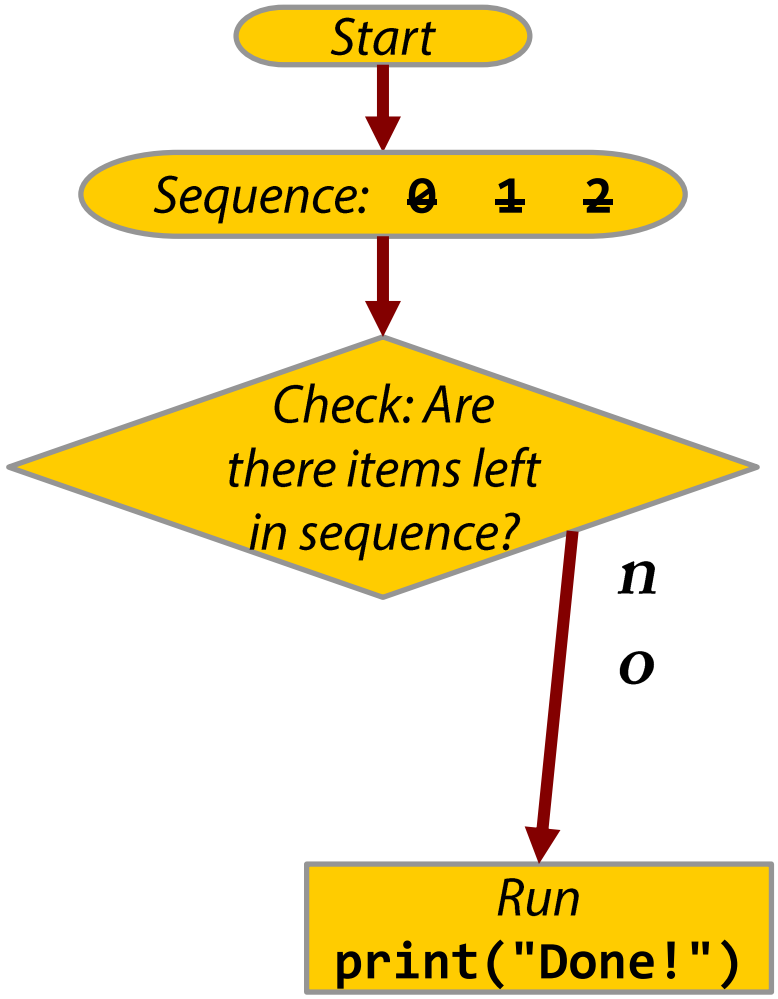
How do `for` loops work?

```
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!")
```



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
```

- *End lecture*

A Closer Look at For Loops

- Typically `for` loops come in two flavors

- **"for each"**
  - Goes through every item in the **sequence**
  - Remember the variable *must be a sequence ("iterable")*

- **Range-based for loop**
  - Goes through every item in a **range**

For Each Loop

- How to write a for each loop:

  for [item] in [sequence]

  - The name of item doesn't matter!

```
word = "program"
for letter in word:
    print(letter)
```

```
word = "program"
for puppy in word:
    print(puppy)
```

For Each Loop

- Iterate through each letter in a string

```
word = "program"
for letter in word:
    print(letter)
```

```
P
R
O
G
R
A
M
```

For Each Loop

- Remember the **"in"** keyword can also be used in if statements

```
word = "abracadabra"
for letter in word:
    if "a" in letter:
        print(letter)
```

```
a
a
a
a
a
```

- **"in"** returns **True** or **False**

For Each Loop

- Will any of these code snippets run?

```
maximum = 10
for number in maximum:
    print(number)
```

```
run = True
for time in run:
    print(time)
```

```
for number in 15:
    print(number)
```

For Each Loop

- A **for each** loop will not run if the variable is not a sequence (*"iterable"*)
- Sequences (iterable types):
  - Strings
  - Lists, Dictionaries (we will cover these later)
- Non-sequences (non-iterable types):
  - Boolean
  - Int, Float

Range-based For Loop

- Range-based for loops require an integer-based range to iterate through

- Similar to a **For Each** loop, the name you give the iterable item does not matter

```
for num in range(10)
for dog in range(10)
for cat in range(10)
```

Range-based For Loop

- Multiple ways to declare a range-based for loop
  - The loop can take 1, 2, or 3 parameters

- Same loop, different declaration:

```
for num in range(10)
for num in range(0, 10)
for num in range(0, 10, 1)
```

Range-based For Loop Can Take Different Parameters

- Single parameter
  - Range ending value
- Two parameters
  - Range starting value, ending value
- Three parameters
  - Range starting value, ending value, and increment
    - Positive increment moves forward
    - Negative increment moves backwards

Range-based For Loop

```
for num in range(5, 0, -1):
        print(num, end=" ")

print("\n Next loop! ")

for num in range(0, 5, 1):
        print(num, end=" ")

print("\n Last loop!")

for num in range(0, 10, 5):
        print(num, end=" ")
```

Range-based For Loop

```
for num in range(5, 0, -1):
        print(num, end=" ")

print("\n Next loop! ")

for num in range(0, 5, 1):
        print(num, end=" ")

print("\n Last loop!")

for num in range(0, 10, 5):
        print(num, end=" ")
```

```
5 4 3 2 1
 Next loop!
0 1 2 3 4
 Last loop!
0 5
```

Be aware of where the loop ends

Range-based For Loop

- Which of these loops are valid?

```
1. for num in range(-1)
2. for num in range(1, 5, -1)
3. for num in range(-1, -5, -1)
4. for num in range(5, 1, 1)
```

Range-based For Loop

- Which of these loops are valid?

```
1. for num in range(-1)
2. for num in range(1, 5, -1)
3. for num in range(-1, -5, -1)
4. for num in range(5, 1, 1)
```

Nested For Loops

- You can use a **for** loop inside a **for** loop!

```
for a in range(3):
  for b in range(3):
    print( str(a) + " " + str(b) )
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

- Do you see a pattern?

Nested For Loops

```
for a in range(3):
  for b in range(3):
    print( str(a) + " " + str(b) )
```

```
0 0
0 1
0 2
1 0
1 1
1 2
2 0
2 1
2 2
```

- Output of 9 lines
  - Two loops running 3 times each: 3 x 3 = 9
- **Loop a** runs only once
- For each and every number in **loop a**, **loop b** is called (starts and ends completely)
  - Therefore, **loop b** runs 3 times

FOR 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*"
- In this class, do **not** use break and continue
  - Though sometimes useful, they can lead to poor understanding of loops

Example

```
for num in range(1, 11, 1):

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

    # skip 5
    if num == 5:
        continue

    print(count)
```

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