# JamCoders: Week 1

Lecture 2B:

- For loops
- Range
- Lists

# Loops

# Repetition

So far, how have we gotten our program to do something over and over?

```
print("Hello world")
print("Hello world")
print("Hello world")
```

```
> Hello World
> Hello World
> Hello World
```

If only there was a better way…

# For Loop

Variable name that gets assigned values from the sequence

Sequence that is read one item at a time

```
for var_name in [sequence]:
    execute_code1()
    ...
```

Codeblock to be repeated

Control structure that runs block of code repeatedly by iterating over elements in sequence.

We call this ***iterating*** *over a sequence*.

# Sequences

# Sequence

For loops need a sequence to iterate over.

This is so common, Python gives us a handy function to create a sequence of numbers...

```
range(start, end, step_size)
```

The *range* function creates a **sequence** of numbers that you can iterate through

# Sequence

The range function can take 1, 2, or 3 arguments:

# Sequence

The range function can take 1, 2, or 3 arguments:

**1 argument**

```
range(end)
```

Counts up starting from 0 to end (exclusive) by 1s

Example:

```
range(6)  # == [0,1,2,3,4,5]
range(2)  # == [0,1]
range(1)  # == [0]
```

# Sequence

The range function can take 1, 2, or 3 arguments:

## 2 arguments

```
range(start, end)
```

Counts up starting from start (inclusive) to end (exclusive) by 1s

Example:

```
range(0, 6)    # == [0,1,2,3,4,5]
range(2, 6)    # == [2,3,4,5]
range(5, 10)   # == [5,6,7,8,9]
```

# Sequence

The range function can take 1, 2, or 3 arguments:

**3 arguments**

```
range(start, end, step_size)
```

Creates a sequence, starting from start, going up to end, in increments of step_size.

Example:

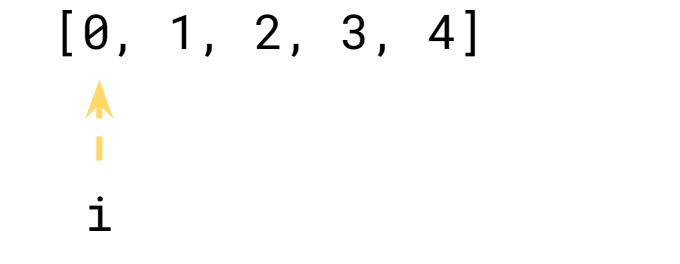
```
range(0, 6, 1)    # == [0,1,2,3,4,5]
range(2, 6, 1)    # == [2,3,4,5]
range(0, 10, 2)   # == [0,2,4,6,8]
range(0, 10, 2)   # == [0,2,4,6,8]
range(5, 0, -1)   # == [5,4,3,2,1]
```

# Range

Let’s see it in action...

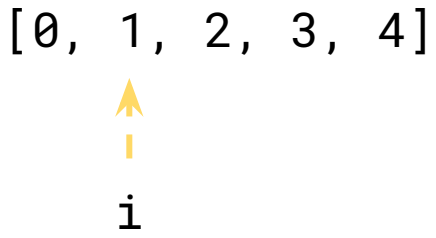
```
for i in range(5):
    print(i)
```

# Range




```
for i in range(5):
    print(i)
```

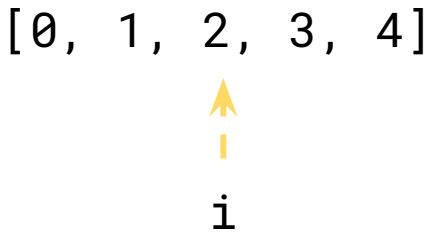
# Range




```
for i in range(5):
    print(i)
```

```
> 0
> 1
```

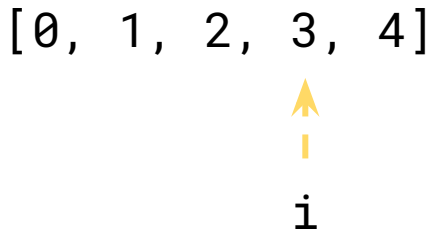
# Range




```
for i in range(5):
    print(i)
```

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

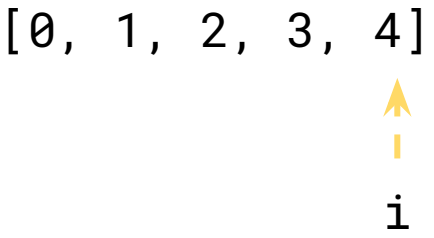
# Range




```
for i in range(5):
    print(i)
```

```
> 0
> 1
> 2
> 3
```

# Range



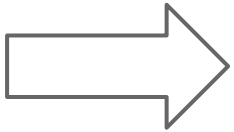

```
for i in range(5):
    print(i)
```

> 0
> 1
> 2
> 3
> 4

# Range

Let’s see it in action...side by side

```
print("Hello world")
print("Hello world")
print("Hello world")
```



(range(3) == [0,1,2])

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

Notice: you don’t have to use the variable if you don’t need it! You can use the range to control how many times the body is executed.

# Sequence

`range`s are not the only kind of sequence...

Where else have you heard “sequence” before in this class?

# Sequence

Where else have you heard “sequence” before in this class?

## Strings

- A *character* is a single letter or symbol, like “c” or “!”
- A *string* is a sequence of characters

In Python, anything within quotation marks is a string and is taken literally:

```
x = "hello"
```

You can use double (“ ”) or single (’ ’) quotation marks, just be consistent.

# Sequence

Strings are a sequence of *characters*!

```
for char in "Howard":
    print(char)
```

```
> H
> o
> w
> a
> r
> d
```

# More Loop Details

Python has 2 special statements you can use inside of loops for finer control.

# Break

The `break` statement tells Python to “break” the loop, and continue on past the for loop body.

```
for i in range(6):
    if i == 3:
        break
    print("Num is", i)
print("La commedia e finita!")
```

```
> Num is 0
> Num is 1
> Num is 2
> La commedia e finita!
```

# Continue

The `continue` statement tells Python to end *just the current iteration* of the loop, and move on to the next value in the sequence.

```
for i in range(6):
    if i == 3:
        continue
    print("Num is", i)
print("La commedia e finita!")
```

```
> Num is 0
> Num is 1
> Num is 2
> Num is 4
> Num is 5
> La commedia e finita!
```

# Live Coding Demo

# Lists

# Lists

Range() gives you a sequence of numbes…

A string is a sequence of characters…

What if you wanted a custom sequence of your own data?

…Lists.

This is a new type! Booleans, Integers, Floats, Strings, + Lists

(Technically, this is your first “data structure”: a way of organizing data. )

# Motivation

Let’s say we want to store a user’s top 10 favorite soccer players

```
player1 = "Ronaldo"
player2 = "Messi"
player3 = "Drogba"

...
```

Each player requires its own variable. This gets more and more cumbersome the more data we have. It’s also hard to answer basic questions, like how many players we have, the first player alphabetically, etc...

# Creating a List

A **list** is an ordered collection of items of any type.

Create a new list using brackets, with a comma-separated list of values.

```
players = ["Ronaldo", "Messi", "Drogba"]
```

# Creating a List

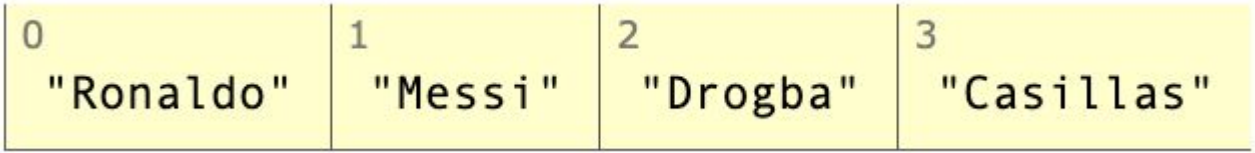
To make an empty **list**, use just empty brackets.

```
players = []
```

# Accessing a List

We use brackets to access a List's values.

```
players = ["Ronaldo", "Messi", "Drogba", "Casillas"]
print(players[0])  # prints "Ronaldo"
print(players[1])  # prints "Messi"
print(players[-1])  # prints "Casillas"
```




Look familiar? Think back to strings! A list is a type of sequence, just like a string.

# Printing Lists

You can use **print** just as you're used to with lists to print out their content.

```
players = ["Ronaldo", "Messi", "Drogba"]
print(players)  # prints ["Ronaldo", "Messi", "Drogba"]
print(players[1:])  # prints ["Messi", "Drogba"]
```

# Iterating over Lists

You can use the same `for x in sequence:` syntax we learned earlier, because lists are a type of sequence just like strings and range.

```
players = ["Ronaldo", "Messi", "Drogba", "Casillas"]
for name in players:
    print(name)
```

Prints each player's name on a new line

# List Functions

# List Appending, Inserting, and Length

- Use `list_name.append(element)` to **append** to the end of the list.
- Use `list_name.insert(index, element)` to **insert** at a specific index.
- Use `len(list_name)` to to get the **length** of the list.

```
items = []
items.append(7)
items.append("Dindu")
items.insert(0, True)
print(items)  # [True, 7, 'Dindu']
print("There are", len(items), "items in the list.")
```

# Removing from a List

To remove an element from the list…

- Use list_name.remove(item) to remove a *specific item*
- Use list_name.pop(index) to remove an item at a *specific index*

```
items = [7, "Dindu", True]
items.pop(1)  # Removes and return "Dindu"
print(items)  # Prints [7, True]
items.remove(7)
print(items)  # Prints [True]
```

# List Contains

To know whether a list contains a particular item, use the `in` keyword

```
ta_list = ["Deontae", "Jonathan", "Dindu", "Bradon"]

dindu_is_ta = "Dindu" in ta_list  # True
vijay_is_ta = "Vijay" in ta_list  # False
```

# List Slicing

Slicing works for lists just like for strings (because lists and strings are both types of sequences, and slicing works for all sequences)!

```
tas = ["Deontae", "Jonathan", "Dindu", "Bradon"]

all_but_one_tas = tas[1:]
print(all_but_one_tas)  # ['Jonathan', 'Dindu', 'Bradon']
```

Slicing makes a (shallow) copy of the array.

# Rapid Fire List Functions

```
numbers = [5, 6, 12, 9, 4]

# Find largest number in list
max_num = max(numbers)  # == 12

# Find smallest number in list
min_num = min(numbers)  # == 4

# Find the sum of numbers in list
total = sum(numbers)  # == 36

fruits = ["apple", "banana", "orange"]
# Find the index of a specific value
banana_index = fruits.index("banana")  # == 1
```

# Coding Demo