

# Lists & The for Loop

C-START Python PD Workshop

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [1, 2, "Hello", "Python"]
```

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [1, 2, "Hello", "Python"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[2]` is ?

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [1, 2, "Hello", "Python"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[2]` is `"Hello"`

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [1, 2, "Hello", "Python"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[2]` is "Hello"
- `my_list[0]` is ?

Lists are a data structure which allow us to store **ordered data**. We can specify list literals in Python using brackets.

```
my_list = [1, 2, "Hello", "Python"]
```

Specific members of a list can be accessed by specifying the **zero-indexed** offset in brackets.

- `my_list[2]` is "Hello"
- `my_list[0]` is 1

# Why Zero-Indexed?

In lower-level programming languages, lists are stored as simply a base address in memory, and the value in the brackets is the offset. The offset is added to the base address to find the memory address of the item.

|                |     |      |      |      |      |      |     |
|----------------|-----|------|------|------|------|------|-----|
| <b>Address</b> | ... | 1350 | 1351 | 1352 | 1353 | 1354 | ... |
| <b>Value</b>   | ... | 12   | 13   | 7    | 21   | 3    | ... |

**Note:** These addresses and values are just an example, not real values.

# List Indexed Assignment

Lists can be changed once they are created, to do so, assign to the list at the index desired.

```
mynums = [4, 5, 6]
mynums[2] = 7
print(mynums)
```

---

```
[4, 5, 7]
```



# List Indexed Assignment

Lists can be changed once they are created, to do so, assign to the list at the index desired.

```
mynums = [4, 5, 6]
mynums[2] = 7
print(mynums)
```

---

```
[4, 5, 7]
```

However, assignment to indices not currently in the list is **not allowed**. This example **will cause an error**.

```
mynums = [4, 5, 6]
mynums[3] = 7      # bad, 3 is out of range!
```

# List Concatenation

Similar to how strings can be concatenated using the + operator, so can lists.

```
a = [5, 6, 7, 8]
b = ["we", "can", "concatenate"]
c = a + b
print(a)
print(b)
print(c)
```

---

```
[5, 6, 7, 8]
```

```
["we", "can", "concatenate"]
```

```
[5, 6, 7, 8, "we", "can", "concatenate"]
```

# Lists in Lists

Lists can store data of **any** type, *including lists*.

```
nested = [[1, 2, 3], [4, 5, 6]]
```

# Lists in Lists

Lists can store data of **any** type, *including lists*.

```
nested = [[1, 2, 3], [4, 5, 6]]
```

Accessing and assigning is done by using another set of brackets.

```
print(nested[0][0])  
nested[0][0] = nested[1][0]  
print(nested)
```

---

```
1  
[[4, 2, 3], [4, 5, 6]]
```

## Practice: List Manipulation

Open a new program in IDLE and define this variable at the top:

```
magiclist = [[1, 2], [3, 4], ["Oh", "Hey"]]
```

Then, your program should (in order):

- 1 Set the second element in the first list to the first element in the first list
- 2 Subtract 1 from the first element in the second list
- 3 Set the second element in the second list to the *length* of the second element in the third list
- 4 Replace the third list with a string obtained by concatenating both elements of the third list together
- 5 Replace the string (the third element of `magiclist`) with its length
- 6 Print `magiclist`

If all went well, your program should print `[[1, 1], [2, 3], 5]`.

# Iterating Over a List using while

Using what we know about `while` loops, we can iterate over a list using a counter variable. Here is an example:

```
i = 0
while i < len(my_list):
    print(my_list[i])
    i = i + 1
```

# Iterating Over a List using while

Using what we know about `while` loops, we can iterate over a list using a counter variable. Here is an example:

```
i = 0
while i < len(my_list):
    print(my_list[i])
    i = i + 1
```

If `my_list` was `[1, 2, "Hello", "Python"]`, then this would print:

```
1
2
Hello
Python
```

# Iterating Over a List using for

Python provides a clean **range-based** construct for iterating over **iterables** called **for**. Here's it's syntax:

```
for var_name in iterable:  
    # do something
```



# Iterating Over a List using for

Python provides a clean **range-based** construct for iterating over **iterables** called **for**. Here's its syntax:

```
for var_name in iterable:  
    # do something
```

So here is an example of iterating over our previous list:

```
for item in my_list:  
    print(item)
```

---

```
1  
2  
Hello  
Python
```

# Generating Ranges

The generator function `range` creates an iterable for looping over a sequence of numbers.

The syntax is `range(start, stop, step)`.

- `start` is the number to start on
- `stop` is the number to stop **before**
- `step` is the amount to increment each time

Here is an example:

```
for i in range(0, 5, 1):  
    print(i)
```

---

0  
1  
2  
3  
4

## Range: step is Optional

If you do not provide `step` to the `range` function, Python will assume that you want to increment by one every time.

Here is an example:

```
for i in range(0, 5):  
    print(i)
```

---

```
0  
1  
2  
3  
4
```

## Range: start is Optional

If you do not provide start or step to the range function, Python will assume that you want to increment by one every time *and* to start at zero.

Here is an example:

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

---

```
0  
1  
2  
3  
4
```

## Repeat $n$ times

So you want to repeat something  $n$  times?

```
n = 5
for i in range(n):
    print("Hello, World!")
```

---

```
Hello, World!
Hello, World!
Hello, World!
Hello, World!
Hello, World!
```

## Practice: Trace the Loops

First, trace the loop by hand and determine the output. Then, type the loop into a Python script and run it to determine if you were correct.

Loop 4

```
for i in range(3):  
    for j in range(2):  
        print(i, j)
```

## Practice: Trace the Loops

First, trace the loop by hand and determine the output. Then, type the loop into a Python script and run it to determine if you were correct.

### Loop 5

```
favnums = [4, 3, 1]
stats = ["my new favorite", "okay", "boring"]
for num in favnums:
    for stat in stats:
        print(num, "is", stat)
    print("Tomorrow...")
print("I'm sticking with", favnums[1])
```