FOR LOOPS

- Loop Basics
- For Loops
- While Loops
- Nested Loops
- Loop Control

A **for loop** will run a specified number of times

- This often corresponds with the length of a list, tuple, or other iterable data type

```
for item in iterable:
```

Indicates a For Loop

A variable name for each item in the iterable

Examples:

- *Price*
- *Product*
- *Customer*

Iterable object to loop through

Examples:

- *List*
- *Tuple*
- *String*
- *Dictionary*
- *Set*

FOR LOOPS

- Loop Basics
- For Loops
- While Loops
- Nested Loops
- Loop Control

A **for loop** will run a specified number of times

- This often corresponds with the length of a list, tuple, or other iterable data type

```
for item in iterable:
    do this
```

Code to run until the loop terminates *(must be indented!)*

Run order:

1. *item = iterable[0]*
2. *item = iterable[1]*
3. *item = iterable[2]*
-
-
- *n. item = iterable[len(iterable)-1]*

FOR LOOPS

- Loop Basics
- For Loops
- While Loops
- Nested Loops
- Loop Control

EXAMPLE *Printing individual letters in a string*

```
iterable = 'Maven'
for item in iterable:
    print(item)
```

```
M ⟵ item = iterable[0]
a ⟵ item = iterable[1]
v ⟵ item = iterable[2]
e ⟵ item = iterable[3]
n ⟵ item = iterable[4]
```

How does this code work?

- Since **'Maven'** is a string, each letter is an **item** we'll iterate through
- **iterable** = ['M', 'a', 'v', 'e', 'n']
- The code will run and **print** each **item** in the **iterable**

FOR LOOPS

- Loop Basics
- For Loops
- While Loops
- Nested Loops
- Loop Control

EXAMPLE *Printing individual letters in a string*

```
word = 'Maven'
for letter in word:
    print(letter)
```

```
M ← letter = word[0]
a ← letter = word[1]
v ← letter = word[2]
e ← letter = word[3]
n ← letter = word[4]
```

How does this code work?

- Since **'Maven'** is a string, we'll iterate through each **letter**
- **word** = ['M', 'a', 'v', 'e', 'n']
- The code will run and **print** each **letter** in the **word**

PRO TIP: Give the components of your loop intuitive names so they are easier to understand

LOOPING OVER ITEMS

Loop Basics

For Loops

While Loops

Nested Loops

Loop Control

Looping over items will run through the items of an iterable

- The loop will run as many times as there are items in the iterable

```
exchange_rate = 0.88
usd_list = [5.99, 9.99, 19.99, 24.99, 99.99]
euro_list = []

for price in usd_list:
    euro_list.append(round(price * exchange_rate, 2))

print(euro_list)
```

```
[5.27, 8.79, 17.59, 21.99, 87.99]
```

*The for loop here is looping over the items, (elements) of **usd_list**, so the loop code block runs 5 times (length of the list):*

1. *price = usd_list[0] = 5.99*
2. *price = usd_list[1] = 9.99*
3. *price = usd_list[2] = 19.99*
4. *price = usd_list[3] = 24.99*
5. *price = usd_list[4] = 99.99*

PRO TIP: To create a new list (or other data type) with loops, first create an empty list, then append values as your loop iterates

LOOPING OVER INDICES

- Loop Basics
- For Loops
- While Loops
- Nested Loops
- Loop Control

Looping over indices will run through a range of integers

- You need to specify a range (usually the length of an iterable)
- This range can be used to navigate the indices of iterable objects

```
exchange_rate = 0.88
usd_list = [5.99, 9.99, 19.99, 24.99, 99.99]
euro_list = []

for i in range(len(usd_list)):
    euro_list.append(round(usd_list[i] * exchange_rate, 2))

print(euro_list)
```

The index is selecting the elements in the list

```
[5.27, 8.79, 17.59, 21.99, 87.99]
```

*The for loop here is looping over indices in a range the size of the **euro_list**, meaning that the code will run 5 times (length of the list):*

1. *i = 0*
2. *i = 1*
3. *i = 2*
4. *i = 3*
5. *i = 4*

PRO TIP: If you only need to access the elements of a single iterable, it's a best practice to loop over items instead of indices

LOOPING OVER MULTIPLE ITERABLES

- Loop Basics
- For Loops
- While Loops
- Nested Loops
- Loop Control

Looping over indices can help with **looping over multiple iterables,** allowing you to use the same index for items you want to process together

EXAMPLE *Printing the price for each inventory item*

```
euro_list = [5.27, 8.79, 17.59, 21.99, 87.99]
item_list = ['Snowboard', 'Boots', 'Helmet', 'Goggles', 'Bindings']

for i in range(len(euro_list)):
    print(f"The {item_list[i].lower()} costs {euro_list[i]} euros.")
```

```
The snowboard costs 5.27 euros.
The boots costs 8.79 euros.
The helmet costs 17.59 euros.
The goggles costs 21.99 euros.
The bindings costs 87.99 euros.
```

*The for loop here is looping over indices in a range the size of the **euro_list**, meaning that the code will run 5 times (length of the list)*

For the first run:

- *i = 0*
- *item_list[i] = Snowboard*
- *euro_list[i] = 5.27*

PRO TIP: ENUMERATE

- Loop Basics
- For Loops
- While Loops
- Nested Loops
- Loop Control

The **enumerate** function will return *both* the index and item of each item in an iterable as it loops through

```
for index, element in enumerate(euro_list):
    print(index, element)
```

```
0 5.27
1 8.79
2 17.59
3 21.99
4 87.99
```

Use the index for multiple lists!

PRO TIP: Use enumerate if you want to loop over an index; it is slightly more efficient and considered best practice, as we are looping over an index derived from the list itself, rather than generating a new object to do so.

```
euro_list = [5.27, 8.79, 17.59, 21.99, 87.99]
item_list = ['Snowboard', 'Boots', 'Helmet', 'Goggles', 'Bindings']

for index, element in enumerate(euro_list):
    print(item_list[index], element)
```

```
Snowboard 5.27
Boots 8.79
Helmet 17.59
Goggles 21.99
Bindings 87.99
```

*We're using the **index** of the euro_list to access each element from the item_list, and then printing each **element** of the euro_list*