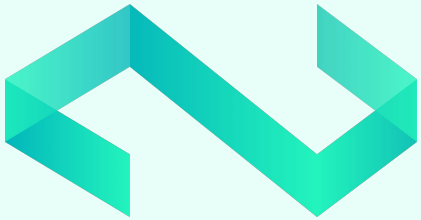
List and For-Loops

CS for Social Good



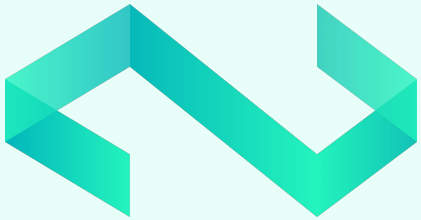
String Indexing

Say we have a string **text**:

text = ‘Hello!’

Question: What if we want to extract the first character in **text**?

We can use **string indexing**, aka getting a character at a certain position (for example, getting the 1st character in **text**, which is **‘H’**).



String Indexing

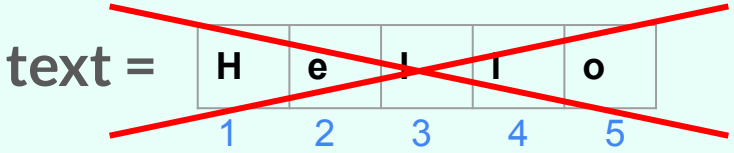
How do we access a specific position? We could simply count through the positions in our string…

text =

H	e	l	l	o
1	2	3	4	5

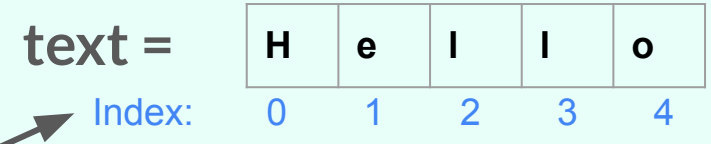
String Indexing

How do we access a specific position? We could simply count through the positions in our string…

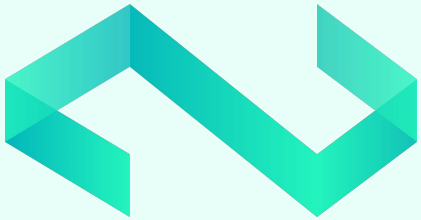


WRONG!!

In Python, there's a concept called **zero-indexing.** In other words, instead of counting our positions from 1, we count from 0.



Note: In computer science, the position is called an **index**



String Indexing

text =	H	e	l	l	o
Index:	0	1	2	3	4

We use square brackets [ ] to access the character at a specific index.

Syntax: `string_name[index]`

For example:

- `text[0]` gives us the first character in text
  - `print(text[0])` prints 'H'
- What about `text[4]`?

String Indexing

text =	H	e	l	l	o
Index:	0	1	2	3	4

We use square brackets [ ] to access the character at a specific index.

For example:

- `text[0]` gives us the first character in text
  - `print(text[0])` prints ‘H’
- `text[4]` gives us the last character in text
  - `print(text[4])` prints ‘o’

String Indexing

There's actually another way to get the last character of a string! Python supports something called **negative indexing**:

text =	H	e	l	l	o
Index:	0	1	2	3	4
Negative index:	-5	-4	-3	-2	-1

For example:

- `text[-1]` gives us the last character in text
  - `print(text[-1])` prints 'o'
- What about `text[-4]`?

String Indexing

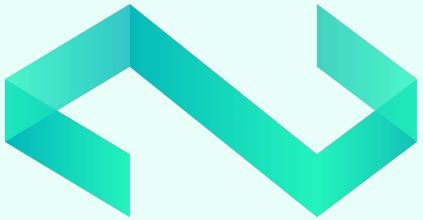
There's actually another way to get the last character of a string! Python supports something called **negative indexing**:

text =	H	e	l	l	o
Index:	0	1	2	3	4
Negative index:	-5	-4	-3	-2	-1

For example:

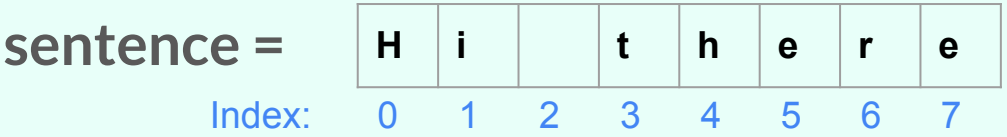
- `text[-1]` gives us the last character in text
  - `print(text[-1])` prints 'o'
- `text[-4]` gives us the fourth-to-last character in text
  - `print(text[-4])` prints 'e'

In general, we only use negative indexing if we want to access the last element, or an element toward the end!



String Slicing

Instead of extracting one character, we can also extract a portion of our string! Let's say we have the variable `sentence = 'Hi there'`



The syntax `sentence[start:end]` gives us a string that begins at the index `start`, and ends at the character before `end` (up to but not including end!)

- `sentence[3:6]` gives us the string `'the'`
  - Notice that it doesn't include the character at index 6!
- `sentence[0:1]` gives us the string `'H'`

String Slicing Syntax

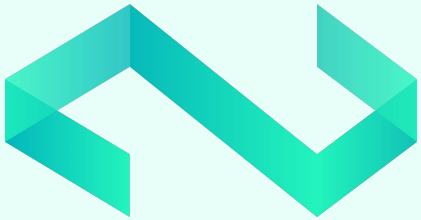
sentence =

H	i		t	h	e	r	e
0	1	2	3	4	5	6	7

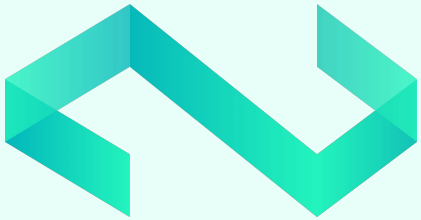
Index:

Some other syntax…

- If we don't include an end index, we go to the end of the string
  - `sentence[3:]` gives us the string `'there'`
- If we don't include a start index, we start from the beginning of the string
  - `sentence[:4]` gives us the string `'Hi t'`
- If start and end index are the same, we get an empty string
  - `sentence[1:1]` gives us the empty string `''`
- If our end index is too big, we go to the end of the string
  - `sentence[3:100]` gives us the string `'there'`



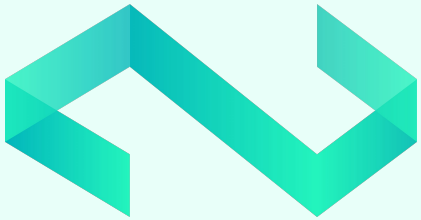
Coding Break



Intro to Data Structures

So far, you've learned how to create and manipulate individual variables.

But what if we want to store a bunch of variables in one place?



Intro to Data Structures

So far, you've learned how to create and manipulate individual variables.

But what if we want to store a bunch of variables in one place?

This is where data structures come in!

[5, 7, 'j', 3, "computer", 900]

Intro to Lists

A list is one of the most basic and useful data structures in Python!

Lists can store all different types of variables (even other lists)! You can add items to lists, remove items from lists, and also access the items in a list through its index.

my_list = [5, 7, ‘j’, 3, “computer”, 900]

0 1 2 3 4 5

Intro to Lists

A list is one of the most basic and useful data structures in Python!

Lists can store all different types of variables (even other lists)! You can add items to lists, remove items from lists, and also access the items in a list through its index.

my_list[0] = 5

my_list = [5, 7, ‘j’, 3, “computer”, 900]

0 1 2 3 4 5

Intro to Lists

A list is one of the most basic and useful data structures in Python!

Lists can store all different types of variables (even other lists)! You can add items to lists, remove items from lists, and also access the items in a list through its index.

my_list[0] = 5

my_list = [5, 7, ‘j’, 3, “computer”, 900]

my_list[2] = ‘j’

0 1 2 3 4 5

Intro to Lists

A list is one of the most basic and useful data structures in Python!

Lists can store all different types of variables (even other lists)! You can add items to lists, remove items from lists, and also access the items in a list through its index.

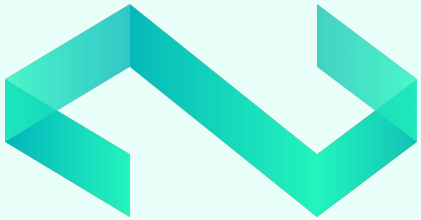
my_list = [5, 7, ‘j’, 3, “computer”, 900]

0 1 2 3 4 5

my_list[0] = 5

my_list[2] = ‘j’

my_list[4] = “computer”



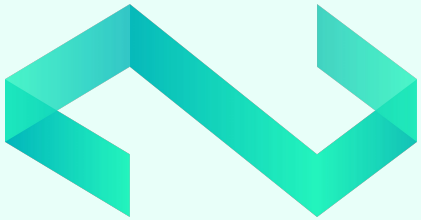
List Methods

There are many functions you can do on lists; today we're just showing you the basics, but feel free to do more research on all the tools at your disposal!

my_list.append(item): add an item to the end of your list

my_list.remove(item): remove & return an item from your list based on its **value**

my_list.pop(index): remove & return an item from your list based on its **index**



```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(3)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

[“Earth”, “Saturn”, “Sun”, “Pluto”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

[“Earth”, “Saturn”, “Sun”, “Pluto”, “Mars”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

[“Earth”, “Saturn”, “Sun”, “Pluto”, “Mars”, “Venus”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

[“Earth”, “Saturn”, “Sun”, “Pluto”, “Mars”, “Venus”, “Moon”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

[“Earth”, “Saturn”, “Sun”, “Pluto”, “Mars”, “Venus”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

[“Earth”, “Saturn”, “Pluto”, “Mars”, “Venus”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

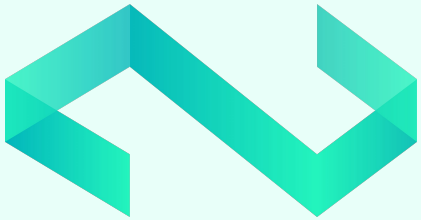
[“Earth”, “Saturn”, “Mars”, “Venus”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

[“Earth”, “Saturn”, “Mars”, “Venus”, “Mercury”]

```
def fun_with_lists():
    planets = [“Earth”, “Saturn”, “Sun”, “Pluto”]
    planets.append(“Mars”)
    planets.append(“Venus”)
    planets.append(“Moon”)
    planets.pop()
    planets.pop(2)
    planets.remove(“Pluto”)
    planets.append(“Mercury”)
    planets.remove(“Saturn”)
```

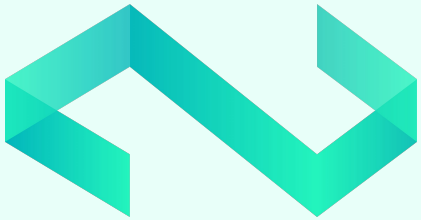
[“Earth”, “Mars”, “Venus”, “Mercury”]



Coding Break

Introducing: For Loops!

(Your new best friend)



Greet everyone

Say, you are tasked with writing a function that greets every person in a list

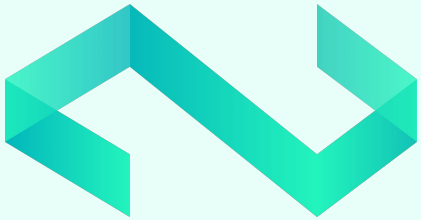
names = [“Alex”, “Jas”, “Jose”, “Luke”]

Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

greet_user(names[0])	Hello Alex!
greet_user(names[1])	Hello Jas!
greet_user(names[2])	Hello Jose!
greet_user(names[3])	Hello Luke!



Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

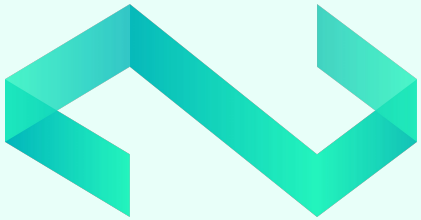
```
def greet_users(names):
        ?
```

Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

```
def greet_users(names):
        greet_user(names[0])
        greet_user(names[1])
        greet_user(names[2])
        ...
```



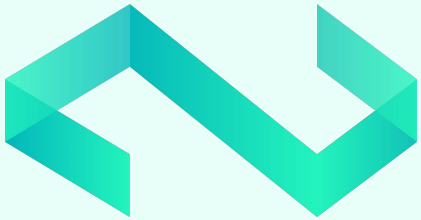
Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

```
def greet_users(names):
    greet_user(names[0])
    greet_user(names[1])
    greet_user(names[2])
    ...
```

- Tedious to type out everything
- Don’t know how long the list will be!



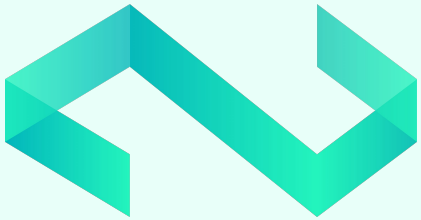
Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

```
def greet_users(names):
        ** what will go here? **
```

- Tedious to type out everything
- Don’t know how long the list will be!



For-loops

A **for-loop** can help you iterate over every element in a list

For-loops

A **for-loop** can help you iterate over every element in a list

```
for element in my_list:
        # do something with element
```

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
        print(element * 2)
print("Done!")
```

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
        print(element * 2)
print("Done!")
```

Output:

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
        print(element * 2)
print("Done!")
```

Output:

element = 2

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
    print("ฅ^•ﻌ•^ฅ")
    print(element * 2)
print("Done!")
```

Output:

ฅ^•ﻌ•^ฅ

element = 2

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
⟹      print(element * 2)
print("Done!")
```

Output:

```
ฅ^•ﻌ•^ฅ
4
```

element = 2

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
        print(element * 2)
print("Done!")
```

Output:

```
ฅ^•ﻌ•^ฅ
4
```

element = 3

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
    print("ฅ^•ﻌ•^ฅ")
    print(element * 2)
print("Done!")
```

Output:

```
ฅ^•ﻌ•^ฅ
4
ฅ^•ﻌ•^ฅ
```

element = 3

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
⟹      print(element * 2)
print("Done!")
```

Output:

```
ฅ^•ﻌ•^ฅ
4
ฅ^•ﻌ•^ฅ
6
```

element = 3

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
        print(element * 2)
print("Done!")
```

Output:

```
ฅ^•ﻌ•^ฅ
4
ฅ^•ﻌ•^ฅ
6
```

element = 4

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
    print("ฅ^•ﻌ•^ฅ")
    print(element * 2)
print("Done!")
```

Output:

```
ฅ^•ﻌ•^ฅ
4
ฅ^•ﻌ•^ฅ
6
ฅ^•ﻌ•^ฅ
```

element = 4

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
⟹      print(element * 2)
print("Done!")
```

element = 4

Output:

```
ฅ^•ﻌ•^ฅ
4
ฅ^•ﻌ•^ฅ
6
ฅ^•ﻌ•^ฅ
8
```

For-loops

A **for-loop** can help you iterate over every element in a list

```
myList = [2, 3, 4]
for element in my_list:
        print("ฅ^•ﻌ•^ฅ")
        print(element * 2)
print("Done!")
```

Output:

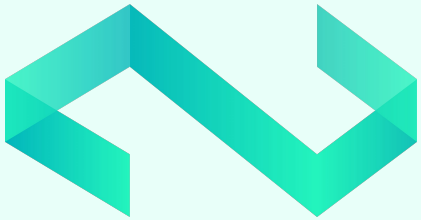
```
ฅ^•ﻌ•^ฅ
4
ฅ^•ﻌ•^ฅ
6
ฅ^•ﻌ•^ฅ
8
Done!
```

Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

def greet_users(names):



Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

```
def greet_users(names):
    for name in names:
        greet_user(name)
```

Greet everyone

Say, you are tasked with writing a function that greets every person in a list

names = [“Alex”, “Jas”, “Jose”, “Luke”]

```
def greet_users(names):
    for name in names:
        greet_user(name)

greet_users(names)
```

```
Hello Alex!
Hello Jas!
Hello Jose!
Hello Luke!
```

For-loops

After testing, I found out these two actually don't give errors in python!! Just gives whatever the value of the variable is at the end of the loop.
Another reason I don't like python >:(

Variables declared within a loop only lives inside the loop

```
myList = [2, 3, 4]
for element in myList:
        print("ฅ^•ﻌ•^ฅ")
        print(element * 2)
print(element)
```

For-loops

Variables declared within a loop only lives inside the loop

```
myList = [2, 3, 4]
for element in myList:
    x = 1
    print(x)
```

Output:

```
1
1
1
```

For-loops

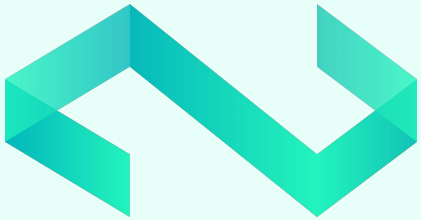
Variables declared within a loop only lives inside the loop

```
myList = [2, 3, 4]
for element in myList:
        x = 1
print(x)
```

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```



For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

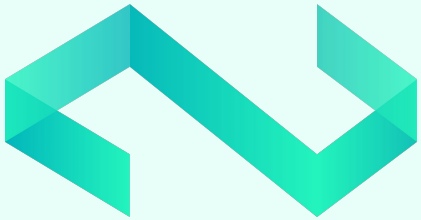
The variable i is going to work just like element did in for-each loops.

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).



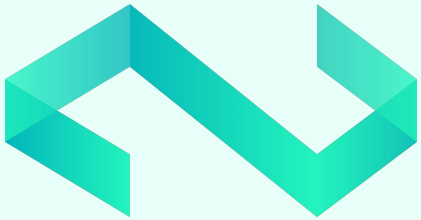
For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

range(5) →



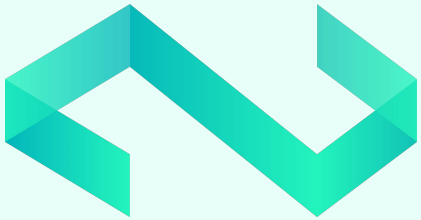
For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

range(5) → [0,1,2,3,4]



For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

range(5) → [0,1,2,3,4]
range(10) →

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

Output:

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

Output:

0

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

Output:

```
0
1
```

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

Output:

```
0
1
2
```

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

Output:

```
0
1
2
3
```

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

Output:

```
0
1
2
3
4
```

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

For-i-in-range loops

In addition to for-each loops, in programming you will often see for-i-in-range loops. These loops have the form:

```
for i in range(n):
    #code block
```

The variable i is going to work just like element did in for-each loops. range(n) is a function that returns a list with integers from 0-(n-1).

```
for i in [0,1,2,3,4]:
    print(i)
```

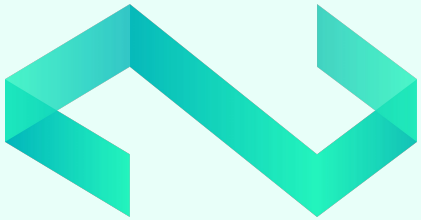
Output:

```
0
1
2
3
4
```

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

Output:

```
0
1
2
3
4
```



Coding Break

Next Time Dictionaries