

CEI.

Python Data Statements

CEI.

if ... else ...

CEI.

if ... else ...

CEI.

- Let's begin to learn about **control flow**
- We often only want certain code to execute when a particular condition has been met.
- For example, **if** my dog is hungry (some condition), then I will feed the dog (some action).

if ... else ...

CEI.

- To control this flow of logic we use some keywords:
 - **if**
 - **elif**
 - **else**

if ... else ...

CEI.

- Control Flow syntax makes use of colons and indentation (whitespace).
- This indentation system is crucial to Python and is what sets it apart from other programming languages.

if ... else ...

CEI.

- Syntax of an **if** statement
if some_condition:
 # execute some code

if ... else ...

CEI.

- Syntax of an **if/else** statement

if some_condition:

 # execute some code

else:

 # do something else

if ... else ...

CEI.

- Syntax of an **if/else** statement

```
if some_condition:
    # execute some code
elif some_other_condition:
    # do something different
else:
    # do something else
```

if ... else ...

CEI.



for loops

CEI.

Many objects in Python are “iterable”, meaning we can iterate over every element in the object.

Such as every element in a list or every character in a string.

We can use for loops to execute a block of code for every iteration.

The term **iterable** means you can “iterate” over the object.

For example you can iterate over every character in a string, iterate over every item in a list, iterate over every key in a dictionary.

- Syntax of a for loop

```
my_iterable = [1,2,3]
for item_name in my_iterable:
    print(item_name)
```

```
>>> 1
```

```
>>> 2
```

```
>>> 3
```

- Syntax of a for loop

```
my_iterable = [1,2,3]
```

```
for item_name in my_iterable:  
    print(item_name)
```

```
>>> 1
```

```
>>> 2
```

```
>>> 3
```


- Syntax of a for loop

```
my_iterable = [1,2,3]
for item_name in my_iterable:
    print(item_name)
```

```
>>> 1
```

```
>>> 2
```

```
>>> 3
```

- Syntax of a for loop

```
my_iterable = [1,2,3]
for item_name in my_iterable:
    print(item_name)
```

```
>>> 1
```

```
>>> 2
```

```
>>> 3
```

- Syntax of a for loop

```
my_iterable = [1,2,3]
for item_name in my_iterable:
    print(item_name)
```

```
>>> 1
```

```
>>> 2
```

```
>>> 3
```

- Syntax of a for loop

```
my_iterable = [1,2,3]
for item_name in my_iterable:
    print(item_name)
```

```
>>> 1
```

```
>>> 2
```

```
>>> 3
```



while loops

CEI.

While loops will continue to execute a block of code **while** some condition remains True.

For example, **while** my pool is not full, keep filling my pool with water.

Or **while** my dogs are still hungry, keep feeding my dogs.

While loops will continue to execute a block of code **while** some condition remains True.

For example, **while** my pool is not full, keep filling my pool with water.

Or **while** my dogs are still hungry, keep feeding my dogs.

- Syntax of a while loop

```
while some_boolean_condition:  
    #do something
```

- You can combine with an else if you want

```
while some_boolean_condition:
```

```
    #do something
```

```
else:
```

```
    #do something different
```



list comprehensions

CEI.

list comprehensions

CEI.

List comprehension is a way to build a new list by applying an expression to each item in an iterable.

list comprehensions

CEI.

List comprehension is a way to build a new list by applying an expression to each item in an iterable.

```
L = []  
L.append(0)  
L.append(1)  
L.append(4)  
L.append(9)  
L.append(16)  
print(L)  
# Prints [0, 1, 4, 9, 16]
```

list comprehensions

CEI.

List comprehension is a way to build a new list by applying an expression to each item in an iterable.

```
L = []  
L.append(0)  
L.append(1)  
L.append(4)  
L.append(9)  
L.append(16)  
print(L)  
# Prints [0, 1, 4, 9, 16]
```

```
L = []  
for x in range(5):  
    L.append(x**2)  
print(L)  
# Prints [0, 1, 4, 9, 16]
```

list comprehensions

CEI.

List comprehension is a way to build a new list by applying an expression to each item in an iterable.

```
L = []  
L.append(0)  
L.append(1)  
L.append(4)  
L.append(9)  
L.append(16)  
print(L)  
# Prints [0, 1, 4, 9, 16]
```

```
L = []  
for x in range(5):  
    L.append(x**2)  
print(L)  
# Prints [0, 1, 4, 9, 16]
```

```
L = [x**2 for x in range(5)]  
print(L)  
# Prints [0, 1, 4, 9, 16]
```


list comprehensions

CEI.

List comprehension is a way to build a new list by applying an expression to each item in an iterable.

```
[expression for var in iterable]
```

Expression

It is evaluated once for each item in iterable

Var

It takes items from an iterable one by one

Iterable

It's a collection of objects (like a list, tuple etc.)

list comprehensions

CEI.

List comprehension is a way to build a new list by applying an expression to each item in an iterable.

```
[ x**2 | for x in range(5) ]
```

1 2

- 1 The first part collects the results of an expression on each iteration and uses them to fill out a new list.
- 2 The second part is exactly the same as the for loop, where you tell Python which iterable to work on. Every time the loop goes over the iterable, Python will assign each individual element to a variable x.

list comprehensions + if else

CEI.


`[expression for var in iterable if_clause]`

```
# Filter list to exclude negative numbers
vec = [-4, -2, 0, 2, 4]
L = [x for x in vec if x >= 0]
print(L)
# Prints [0, 2, 4]
```

list comprehensions + if else

CEI.

[*expression* for *var* in *iterable*]



[*expression* for *var* in *iterable* for *var* in *iterable*]

list comprehensions + if else

CEI.

```
# With list comprehension
vector = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
L = [number for list in vector for number in list]
print(L)
# Prints [1, 2, 3, 4, 5, 6, 7, 8, 9]
```



dict comprehensions

CEI.

dict comprehensions

CEI.

```
{key:value for var in iterable}
```

key: value

*Key & value can be any
expression & evaluated once
for each item in iterable*

var

*It takes items from an
iterable one by one*

Iterable

*It's a collection of objects
(like a list, tuple etc.)*

dict comprehensions

CEI.

`{key:value for var in iterable}`

key: value

*Key & value can be any
expression & evaluated once
for each item in iterable*

var

*It takes items from an
iterable one by one*

Iterable

*It's a collection of objects
(like a list, tuple etc.)*

```
D = {x: x**2 for x in range(5)}  
print(D)  
# Prints {0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

dict comprehensions

CEI.

```
{ x: x**2 | for x in range(5) }
```

1 2

- 1 The first part collects the key/value results of expressions on each iteration and uses them to fill out a new dictionary.
- 2 The second part is exactly the same as the for loop, where you tell Python which iterable to work on. Every time the loop goes over the iterable, Python will assign each individual element to a variable x.

dict comprehensions + if else

CEI.

`{key:value for var in iterable if_clause}`

```
D = {x: x**2 for x in range(6) if x % 2 == 0}
```

```
print(D)
```

```
# Prints {0: 0, 2: 4, 4: 16}
```

dict comprehensions + if else

CEI.

```
{key:{dict comprehension} for var in iterable}
```

```
D = {(k,v): k+v for k in range(2) for v in range(2)}  
print(D)  
# Prints {(0, 1): 1, (1, 0): 1, (0, 0): 0, (1, 1): 2}  
  
# is equivalent to  
D = {}  
for k in range(2):  
    for v in range(2):  
        D[(k,v)] = k+v  
print(D)  
# Prints {(0, 1): 1, (1, 0): 1, (0, 0): 0, (1, 1): 2}
```

C.