Hyperiondev

# Workshop - Iterations & Lists

## Lecture – Housekeeping

- ❑ The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
- ❑ No question is daft or silly - **ask them!**
- ❑ There are Q/A sessions midway and at the end of the session, should you wish to ask any follow-up questions.
- ❑ For all non-academic questions, please submit a query: www.hyperiondev.com/support
- ❑ Report a safeguarding incident: http://hyperiondev.com/safeguardreporting
- ❑ We would love your feedback on lectures: https://hyperionde.wufoo.com/forms/zsqv4m40ui4i0q/

## Github Repository – Lecture Examples/Slides

https://github.com/HyperionDevBootcamps/C4_SE_lecture_examples

## PEP documentation

https://peps.python.org/pep-0548/

## Objectives

1. Iteration
   a. While Loops
   b. For loops
2. Lists
   a. List methods
   b. List operations
   c. List comprehension

# Loops

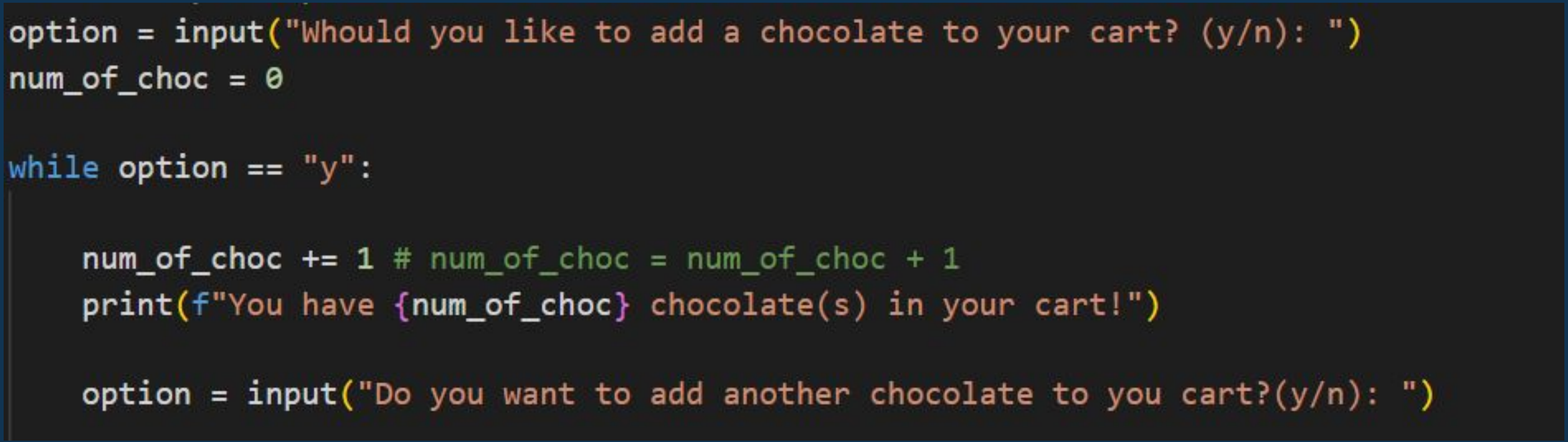
★ Loops are used when we need to repeat a certain block of code multiple times.

★ Remember there are two types of loops:
  - **while** loops
  - **for** loops

# while Loops

- ★ While loops are used in situations when we are not sure how many times we need to repeat the code block.
- ★ Therefore, we can use a while loop to execute a certain condition. While our condition is True, the code within the loop will execute, however, the loop will terminate the moment our condition becomes False.

## while Loop Example and Syntax

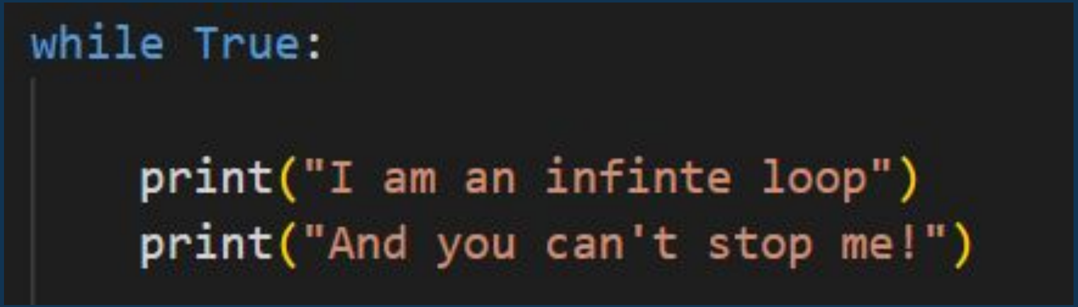


```python
option = input("Whould you like to add a chocolate to your cart? (y/n): ")
num_of_choc = 0

while option == "y":

    num_of_choc += 1 # num_of_choc = num_of_choc + 1
    print(f"You have {num_of_choc} chocolate(s) in your cart!")

    option = input("Do you want to add another chocolate to you cart?(y/n): ")
```

Hyperiondev

## Infinite Loops

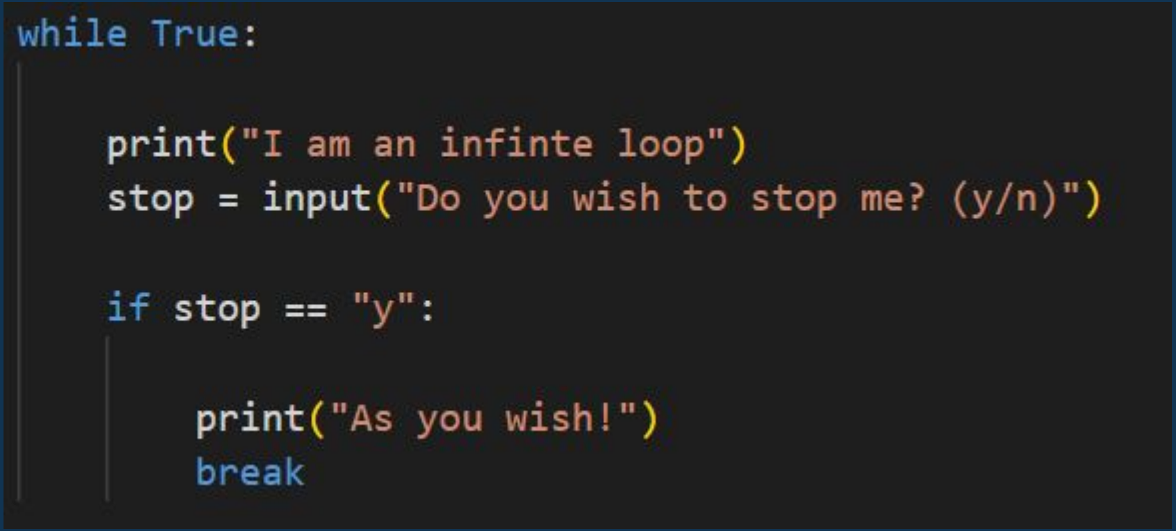
- ★ There may be some cases where we would need the loop to keep looping for as long as the program is running.
- ★ This would be referred to as an infinite loop.
- ★ Example:



```
while True:

    print("I am an infinte loop")
    print("And you can't stop me!")
```

## Breaking the Loop

- ★ At some point, we would like to **break** out of our infinite loop. In order to achieve that, we can use the break statement to exit the loop.
- ★ Example:



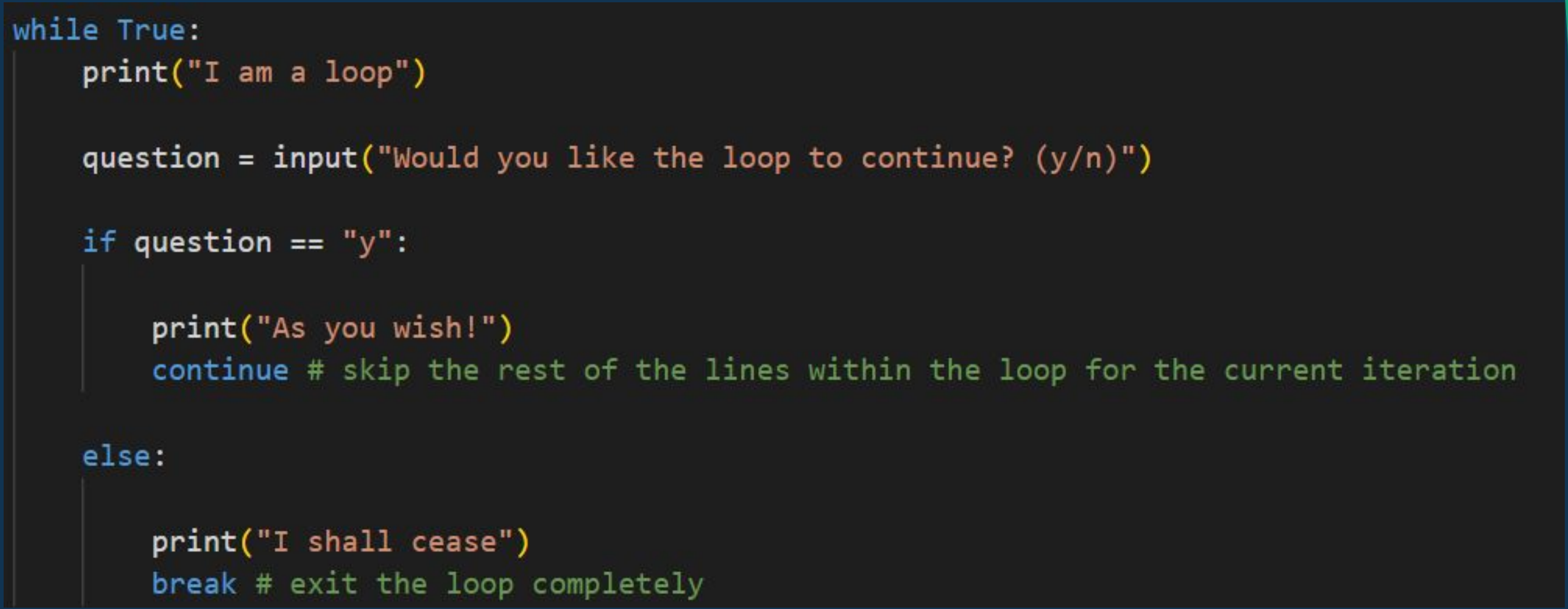
```
while True:

    print("I am an infinte loop")
    stop = input("Do you wish to stop me? (y/n)")

    if stop == "y":

        print("As you wish!")
        break
```

Hyperiondev

## Continuing the Loop

- ★ The continue statement is used to skip any and all lines of code within a loop for the current iteration only.
- ★ The loop will not terminate, but will continue with the next iteration.
- ★ The loop will not break.

# Example: Continuing the loop



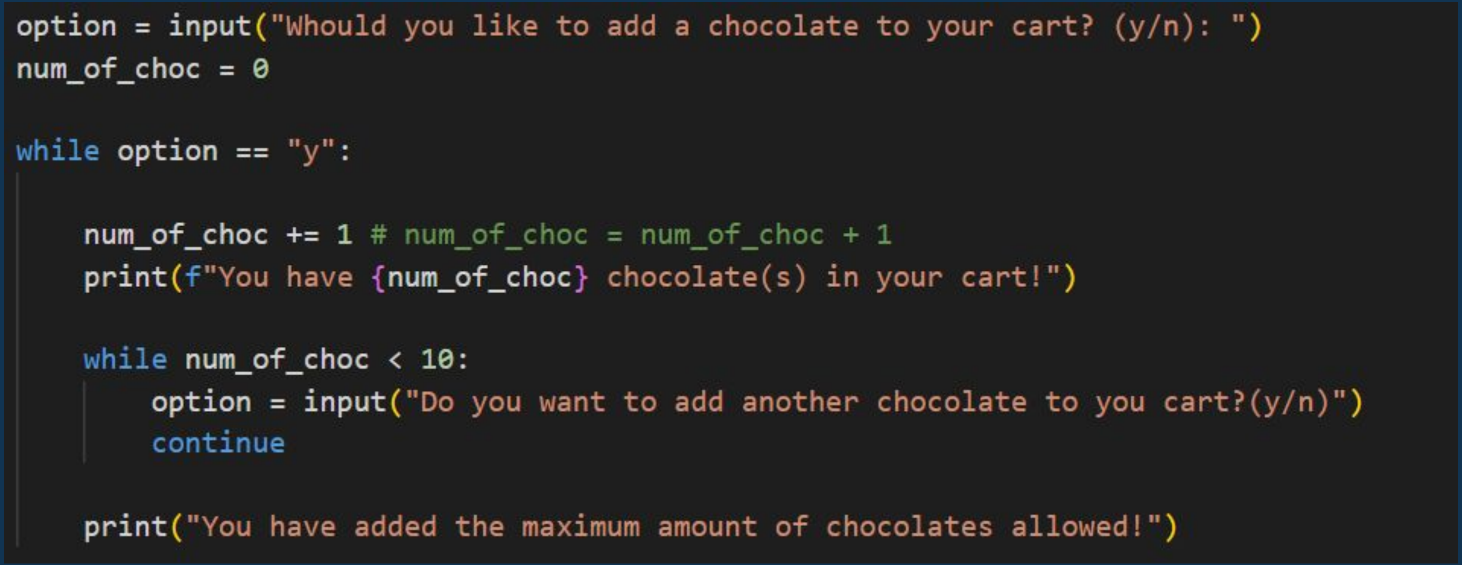
```python
while True:
    print("I am a loop")

    question = input("Would you like the loop to continue? (y/n)")

    if question == "y":

        print("As you wish!")
        continue # skip the rest of the lines within the loop for the current iteration

    else:

        print("I shall cease")
        break # exit the loop completely
```

Hyperiondev

# Nested While Loop

Syntax for a nested while loop:

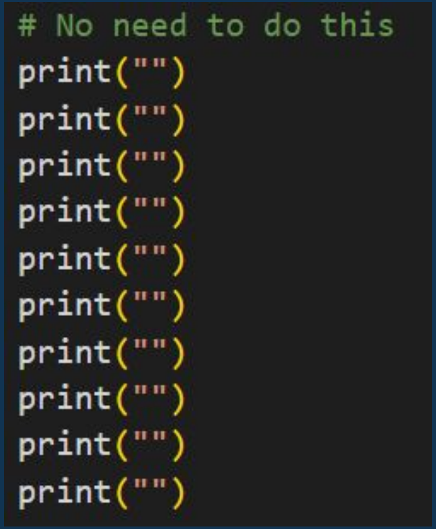
```
while condition:
    while condition:
        statement(s)
    statement(s)
```



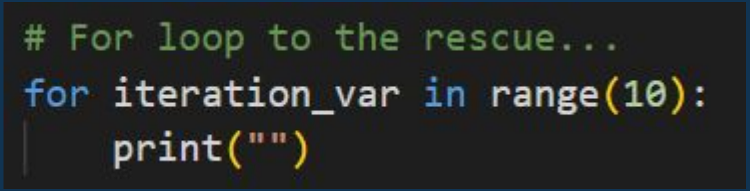
```python
option = input("Whould you like to add a chocolate to your cart? (y/n): ")
num_of_choc = 0

while option == "y":

    num_of_choc += 1 # num_of_choc = num_of_choc + 1
    print(f"You have {num_of_choc} chocolate(s) in your cart!")

    while num_of_choc < 10:
        option = input("Do you want to add another chocolate to you cart?(y/n)")
        continue

    print("You have added the maximum amount of chocolates allowed!")
```

# for Loops

- ★ For loops are used when we need code to run a specified amount of times.
- ★ Think of it making the task of creating ten print statements much easier.

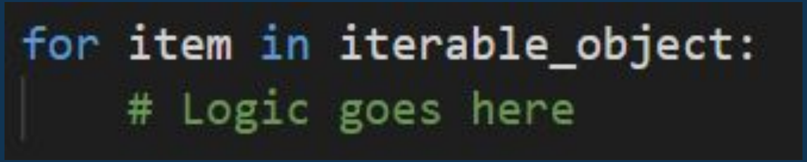


```python
# No need to do this
print("")
print("")
print("")
print("")
print("")
print("")
print("")
print("")
print("")
print("")
```



```python
# For loop to the rescue...
for iteration_var in range(10):
    print("")
```

# for Loop Syntax

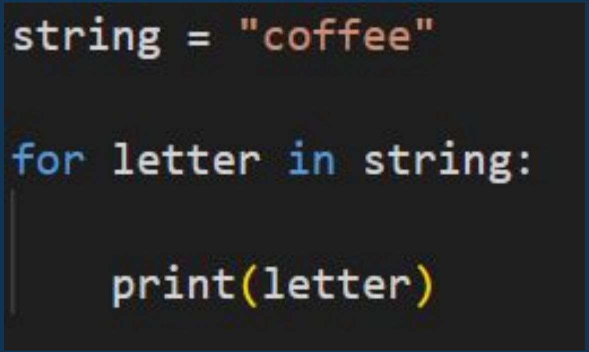


```
for item in iterable_object:
    # Logic goes here
```

★ Iterable_object: a list of numbers, a string of characters, a range etc.

★ Item: temporary variable used inside the for loop to reference the current position of our iterator.

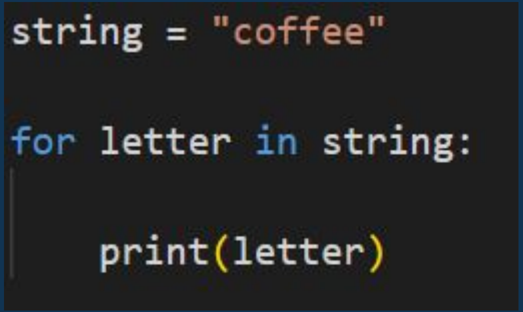
## for Loop Example



```
string = "coffee"

for letter in string:

    print(letter)
```

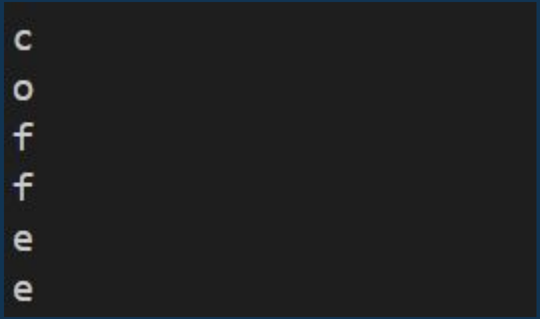
★ The above loop will iterate over the string "coffee".
★ This entails that the temporary variable letter will continuously be updated with each letter found in "coffee".
★ Which results in the following output:

## for Loop Example Cont.



```
string = "coffee"

for letter in string:

    print(letter)
```

**[output]**



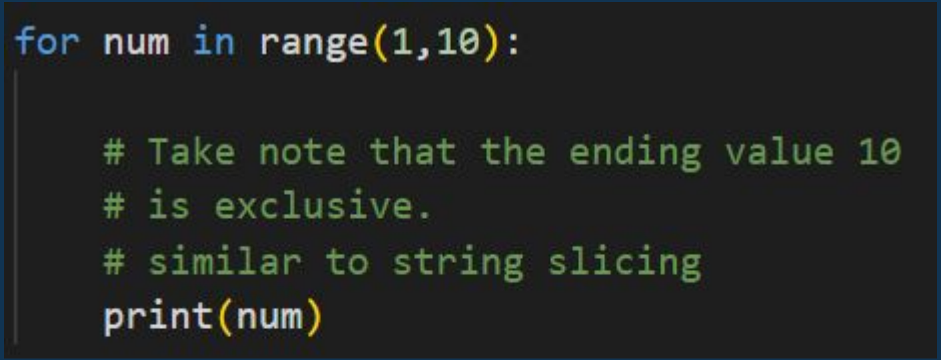
```
c
o
f
f
e
e
```

Since letter will iterate over every instance of string, we get the output of “coffee” spelt on separate lines.

Hyperiondev

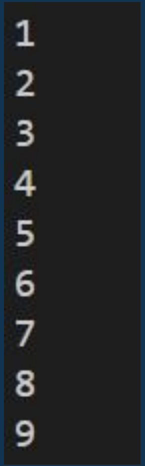
## for Loops and Range

★ With for loops we can also get a range of numbers from a starting value to an ending value.



```
for num in range(1,10):

    # Take note that the ending value 10
    # is exclusive.
    # similar to string slicing
    print(num)
```

**[output]**



```
1
2
3
4
5
6
7
8
9
```

The output here will be all values from 1 to 9.

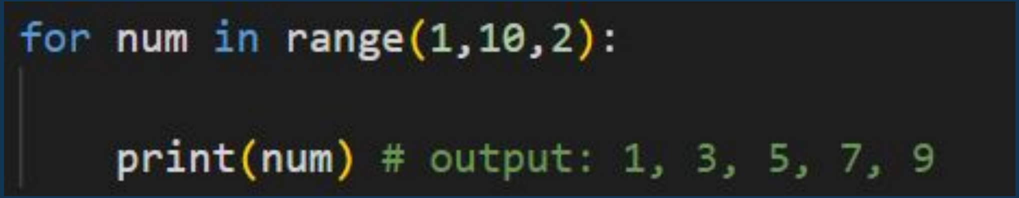
## Range

★ Range allows us to run a block of code a specified amount of times.

| Range | Description | Additional Info |
|---|---|---|
| range(10) | Outputs integers from 0 through 9 | Range will always start from 0 |
| range(1, 10) | Outputs integers from 1 to 9 | Parameters(start, end) |
| range(1, 10, 2) | Outputs odd numbers from 1 to 10 | Third available parameter is “step” (how many to skip) |
| range(10, 1, -1) | Outputs integers from 10 to 1 | Negative counter that skips backwards |

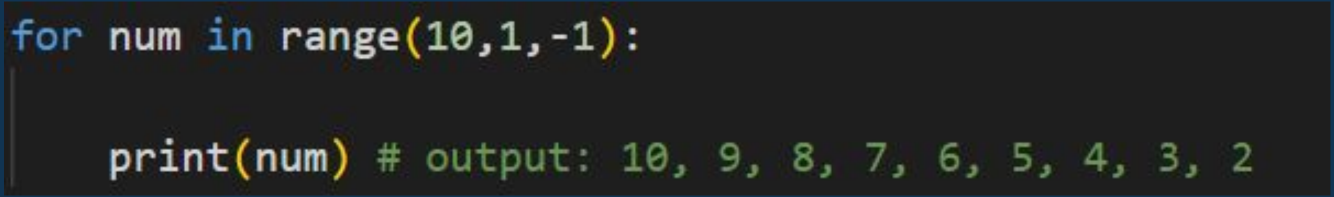
## for Loops and Range

- ★ The third parameters specifies the ‘step’.
- ★ It similar to having in increment variable eg. i +=1



```python
for num in range(1,10,2):

    print(num) # output: 1, 3, 5, 7, 9
```

- ★ If the third parameters is a negative number, it means steps ‘back’.
- ★ It similar to having in increment variable eg. i -=1



```python
for num in range(10,1,-1):

    print(num) # output: 10, 9, 8, 7, 6, 5, 4, 3, 2
```

## Nested for loops

```python
for i in range(0,3):
    for j in range(0,3):
        print(i,j)
```

```
#prints
# 0 0
# 0 1
# 0 2
# 1 0
# 1 1
# 1 2
# 2 0
# 2 1
# 2 2
```

```python
for i in range(1,10):
    for j in range(9,10):
        print(f"{i} x {j} = {i*j}")
```

```
#prints
# 1 x 9 = 9
# 2 x 9 = 18
# 3 x 9 = 27
# 4 x 9 = 36
# 5 x 9 = 45
# 6 x 9 = 54
# 7 x 9 = 63
# 8 x 9 = 72
# 9 x 9 = 81
```

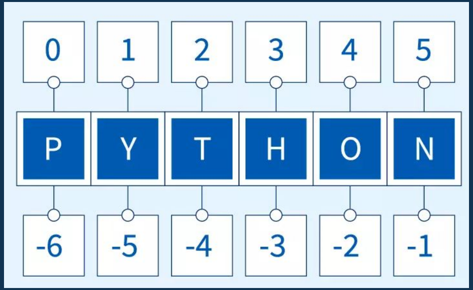
Hyperiondev

# Lists

★ Lists are used when we need to store a lot of data, or the order in which the data is stored is important.

★ Lists are capable of holding many items in one place as well as keeping the data in order.

★ Python will also provide each piece of data an index that represents its position in the list.



## Lists Cont.

- ★ A list is a specialised format of storing and organising data.
- ★ A list is basically a group of items / data.
- ★ Lists are known as sequence data types because they behave like an ordered collection of items.

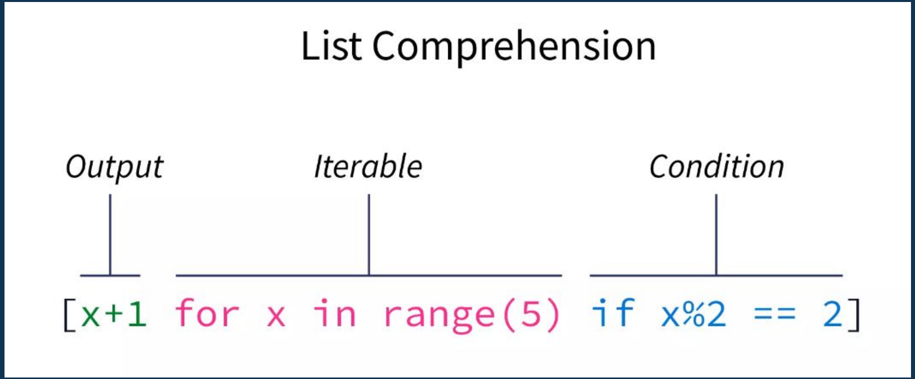
## Methods

- ★ **extend()** - Adds all elements of a list to the another list
- ★ **insert()** - Inserts an item at the defined index
- ★ **remove()** - Removes an item from the list
- ★ **pop()** - Removes and returns an element at the given index
- ★ **index()** - Returns the index of the first matched item
- ★ **count()** - Returns the count of number of items passed as an argument
- ★ **sorted()** - Sorts items in a list in ascending order
- ★ **reverse()** - Reverses the order of items in the list

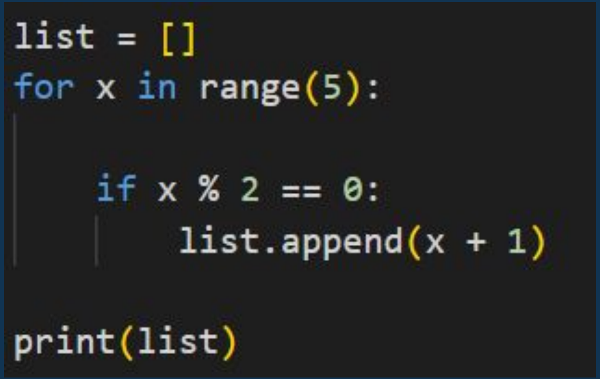
# List operations

- ★ **Creating a list using:** str_list = [“cat”, “dog”, “fish”]
- ★ **Indexing a list:** str_list[0] -> cat
- ★ **Slicing a list:** str_list[0:2] -> [“cat”, “dog”]
- ★ **Changing elements in a list:** str_list[2] = “horse”
  -> [“cat”, “dog”, “horse”]
- ★ **Adding an element to a list:** str_list.append(“hamster”)
  -> [“cat”, “dog”, “horse”, “hamster”]

# List comprehension



```
[x+1 for x in range(5) if x%2 == 2]
```



```
list = []
for x in range(5):

    if x % 2 == 0:
        list.append(x + 1)

print(list)
```

https://www.scaler.com/topics/list-comprehension-in-python/

Hyperiondev

Hyperiondev

## Q & A Section

Please use this time to ask any questions relating to the topic explained, should you have any

Hyperiondev

## Thank you for joining us

Stay hydrated
Avoid prolonged screen time
Take regular breaks
Have fun :)