



# Loops



# Table of Contents



Definitions

while Loop

for Loop

Working with the Iterators

Operations with for Loop

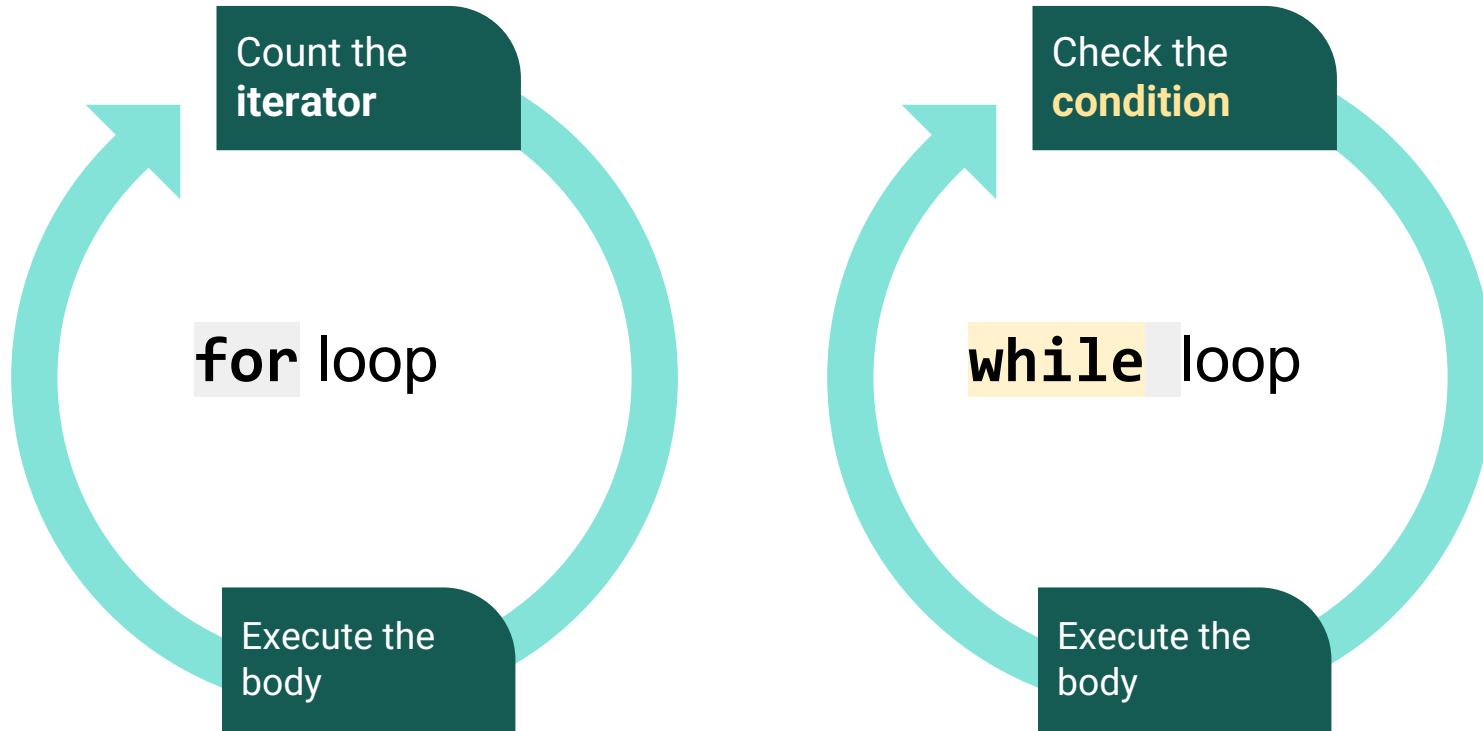
Nested for Loop



# Definitions

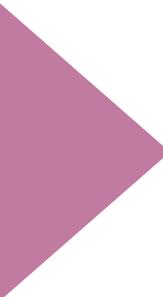
```
for i in iterator :  
    print(i)
```

# Definitions - Loops



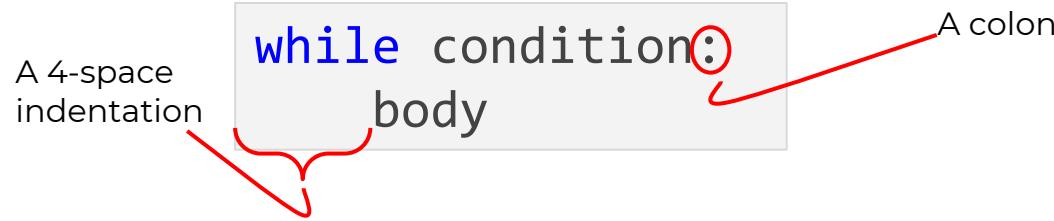


# while Loop



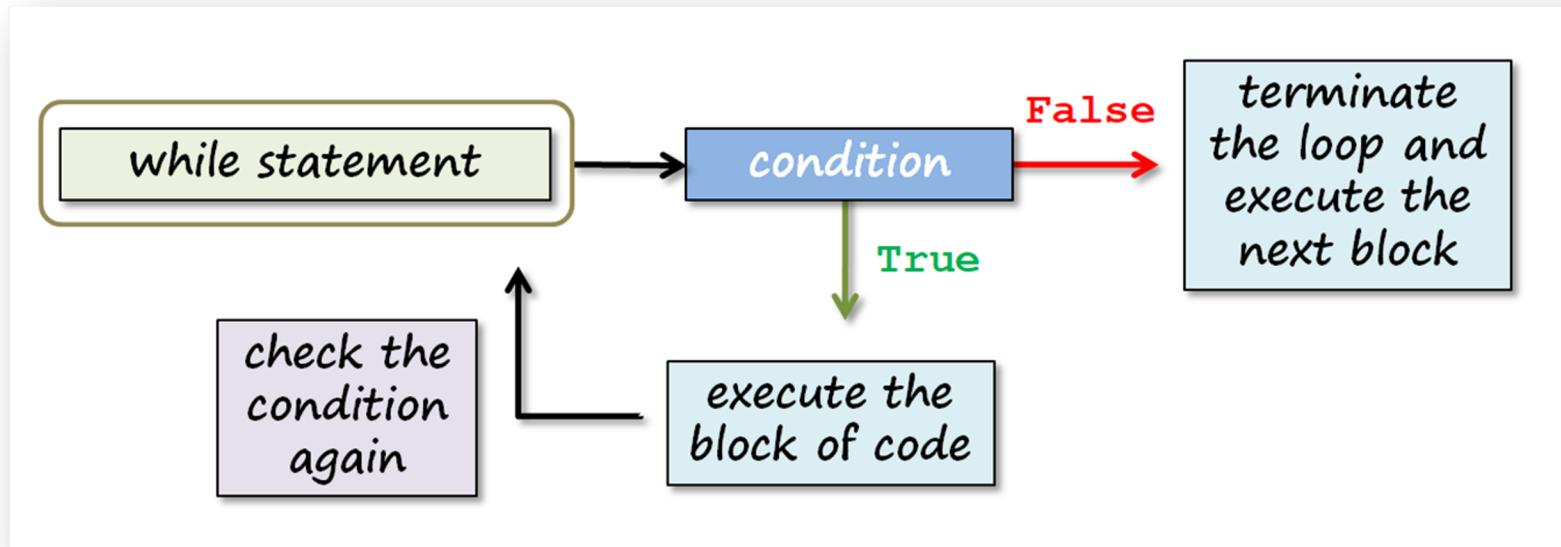
# while Loop (review the pre-class)

- ▶ The simple syntax ↴ of a **while** loop is :



# while Loop (review the pre-class)

- The basic diagram ↗ of a **while** loop works as follows :



# while Loop (review the pre-class)

- Let's take a look at the first `while` loop in the pre-class content :

```
1 number = 0
2
3 while number < 6:
4     print(number)
5     number += 1
6 print('now, number is bigger or equal to 6')
7
```

# while Loop (review the pre-class)

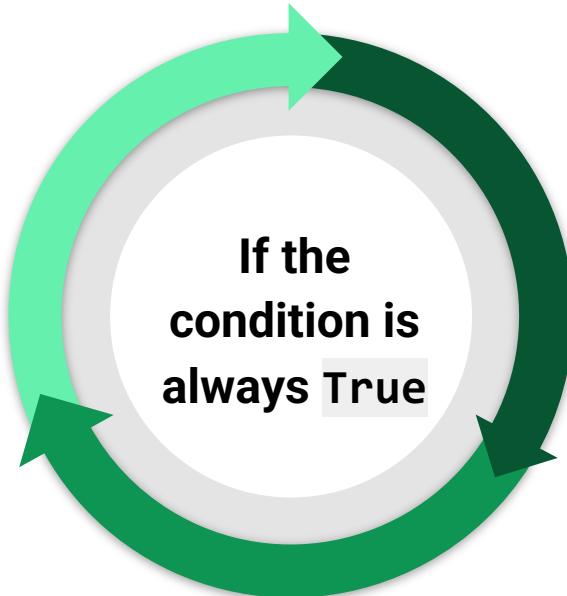
- ▶ The output :

```
1 number = 0
2
3 while number < 6:
4     print(number)
5     number += 1
6 print('now, number is bigger or equal to 6')
7
```

```
1 0
2 1
3 2
4 3
5 4
6 5
7 now, number is bigger or equal to 6
8
```

# while Loop

- ▶ Pay attention not to start an infinite loop.



# while Loop (review the pre-class)

- ▶ We can use an iterator object in a **while** loop. Let's call a **list** in this example :

```
1 my_list=["a", "b", "c", "d", "e"]
2
3 a = 0
4
5 while a < len(my_list):
6     print('square of {} is : {}'.format(a, a**2))
7     a+=1
8
```

# while Loop (review the pre-class)

- ▶ The output :

```
1 my_list=["a", "b", "c", "d", "e"]
2
3 a = 0
4
5 while a < len(my_list):
6     print('square of {} is : {}'.format(a, a**2))
7     a+=1
8
```

```
1 square of 0 is : 0
2 square of 1 is : 1
3 square of 2 is : 4
4 square of 3 is : 9
5 square of 4 is : 16
6
```

# while Loop (review the pre-class)

- ▶ **Task:** Take the age of the user using `input()` and `while` loop.
  - ▶ Write a program that ;
    - ▶ Takes the age from user,
    - ▶ Check the age if it is correct numeric format.

# while Loop (review the pre-class)

- ▶ The code can be like :

```
1 age = input("Enter your age please : ")
2
3 while not age.isdigit():
4     print ("You entered incorrectly!")
5     age = input("Enter your age please : ")
6
7 print("Great! You enter valid input : ", age)
8
```

# while Loop (review the pre-class)

## ► Task:

Let's play famous 'guessing a number game' using **while** loop.

- Write a program that ;
  - Takes the numbers from user,
  - Compares the number the user entered with the number you assigned and then gives a message “Little lower” or “Little higher” till the user knows it.

# while Loop (review the pre-class)

- The code can be like : In this case, we are trying to find number 28.

```
1 answer = 28
2
3 question = 'What a two-digit number am I thinking of? '
4 print ("Let's play the guessing game!")
5
6 while True:
7     guess = int(input(question))
8
9     if guess < answer:
10         print('Little higher')
11     elif guess > answer:
12         print('Little lower')
13     else: # guess == answer
14         print('Are you a MINDREADER!!!')
15         break
16
```

# while Loop

- Lastly, let's play famous 'guessing game' using **while** loop.

We have written a program that does not exit the **while** loop until you find the correct number.

```
1 answer = 28  
2  
3 question = 'What a two-digit number am I thinking of? '  
4 print ("Let's play the guessing game!")  
5  
6 while True:  
7     guess = int(input(question))  
8  
9     if guess < answer:  
10        print('Little higher')  
11    elif guess > answer:  
12        print('Little lower')  
13    else: # guess == answer  
14        print('Are you a MINDREADER!!!')  
15        break  
16
```

When the user knows the answer (28) and enters input, it takes the value of 28 and assigns to variable **guess**, in the end, **else** works and breaks the loop.

We used **break** keyword in order to quit and exit the **while** loop.

# while Loop

## ► Task:

Find and print the length of the **longest word**.

- ▶ Write a program that ;
  - ▶ Takes a **string sentence** consisting of a couple of words from the user,
  - ▶ Compares and find out the **longest** word and **prints** the whole sentence and the **length** of the longest word as **int** type.
  - ▶ Use **while** loop.

# while Loop

- ▶ The code can be like :

```
1 sentence = input("Give me a sentence :")
2 words = sentence.split()
3 i = 0
4 longest = 0
5 while i < len(words) :
6     if len(words[i]) > longest:
7         longest = len(words[i])
8     i += 1
9 print("the length of the longest word :", longest)
10
11
```



# for Loop

the world and  
even life itself are  
nothing but loops.

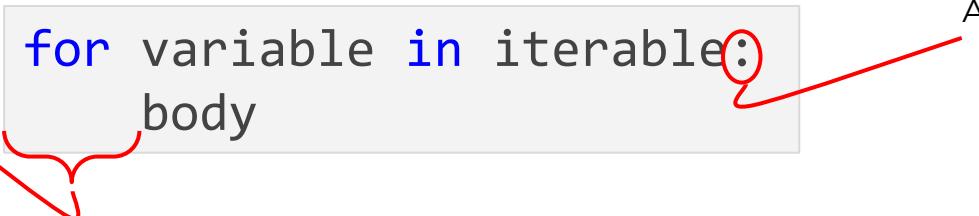
# for Loop (review the pre-class)

- ▶ The simple syntax ↗ of a **for** loop is :

```
for variable in iterable:  
    body
```

A 4-space indentation

A colon



# for Loop (review the pre-class)

- ▶ To create a **for** loop, you need a variable and an iterable object.
- ▶ Let's examine the subject through an example :

```
1 for i in [1, 2, 3, 4, 5] :  
2     print(i)  
3
```

# for Loop (review the pre-class)

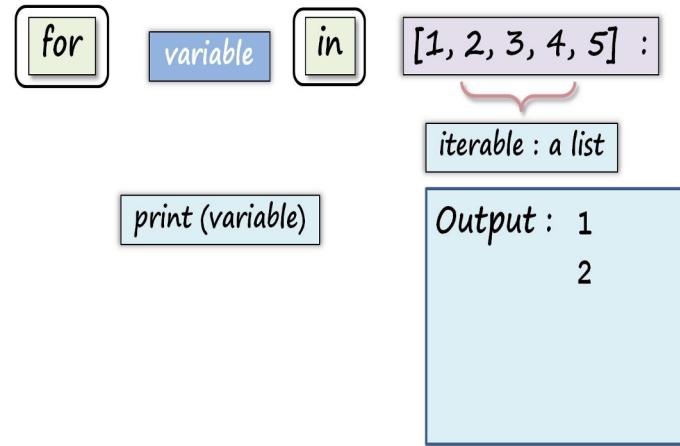
- ▶ To create a **for** loop, you need a variable and an iterable object.
- ▶ Let's examine the subject through an example :

```
1 for i in [1, 2, 3, 4, 5] :  
2     print(i)  
3
```

```
1 1  
2 2  
3 3  
4 4  
5 5  
6
```

# for Loop (review the pre-class)

- You can follow the animated diagram of this **for** loop for a better understanding.



# for Loop (review the pre-class)

- ▶ Another example :

```
1 seasons = ['spring', 'summer', 'autumn', 'winter']
2
3 for season in seasons :
4     print(season)
5
```

# for Loop (review the pre-class)

- ▶ In the structure of the **for** loop, you can use also a variable as an iterable.
- ▶ Let's see it in an example :

```
1 seasons = ['spring', 'summer', 'autumn', 'winter']
2
3 for season in seasons :
4     print(season)
5
```

```
1 spring
2 summer
3 autumn
4 winter
5
```

# for Loop

- In the structure of the **for** loop, you can use also a variable as an iterable.
- Let's see it in an example :

```
1 seasons = ['spring', 'summer', 'autumn', 'winter']
2
3 for season in seasons :
4     print(season)
5
```

```
1 spring
2 summer
3 autumn
4 winter
5
```

# for Loop

- In the structure of the `for` loop, you can use also a variable as an iterable.
- Let's see it in an example :

```
1 seasons = ['spring', 'summer', 'autumn', 'winter']
2
3 for season in seasons :
4     print(season)
5
```

```
1 spring
2 summer
3 autumn
4 winter
5
```

# for Loop

- In the structure of the `for` loop, you can use also a variable as an iterable.
- Let's see it in an example :

```
1 seasons = ['spring', 'summer', 'autumn', 'winter']
2
3 for season in seasons :
4     print(season)
5
```

```
1 spring
2 summer
3 autumn
4 winter
5
```

# for Loop

- In the structure of the `for` loop, you can use also a variable as an iterable.
- Let's see it in an example :

```
1 seasons = ['spring', 'summer', 'autumn', 'winter']
2
3 for season in seasons :
4     print(season)
5
```

```
1 spring
2 summer
3 autumn
4 winter
5
```

# for Loop

## ▶ Task : Python Program to say “hello name”

- ▶ Write a program to say “hello names” from the following list.
- ▶ Print the result such as : “hello Samuel”  
“hello Victor”

```
names = ["Ahmed", "Aisha", "Adam", "Joseph", "Gabriel"]
```

# for Loop

- ▶ The code might be like :

```
1 names = ["Ahmed", "Aisha", "Adam", "Joseph", "Gabriel"]
2
3 for i in names:
4     print("hello", i)
5
```

## Output

```
hello Ahmed
hello Aisha
hello Adam
hello Joseph
hello Gabriel
```

# for Loop

- ▶ **Task : Python Program to create numbers using range()**
  - ▶ Write a program to create a **list** consisting of numbers from **1** to **5**.
  - ▶ Print the result such as : [1, 2, 3, 4, 5]

# for Loop

- ▶ The code might be like :

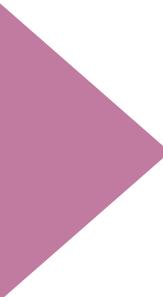
```
1 numbers = []
2
3 for i in range(1, 6):
4     numbers.append(i)
5 print(numbers)
6
```

## Output

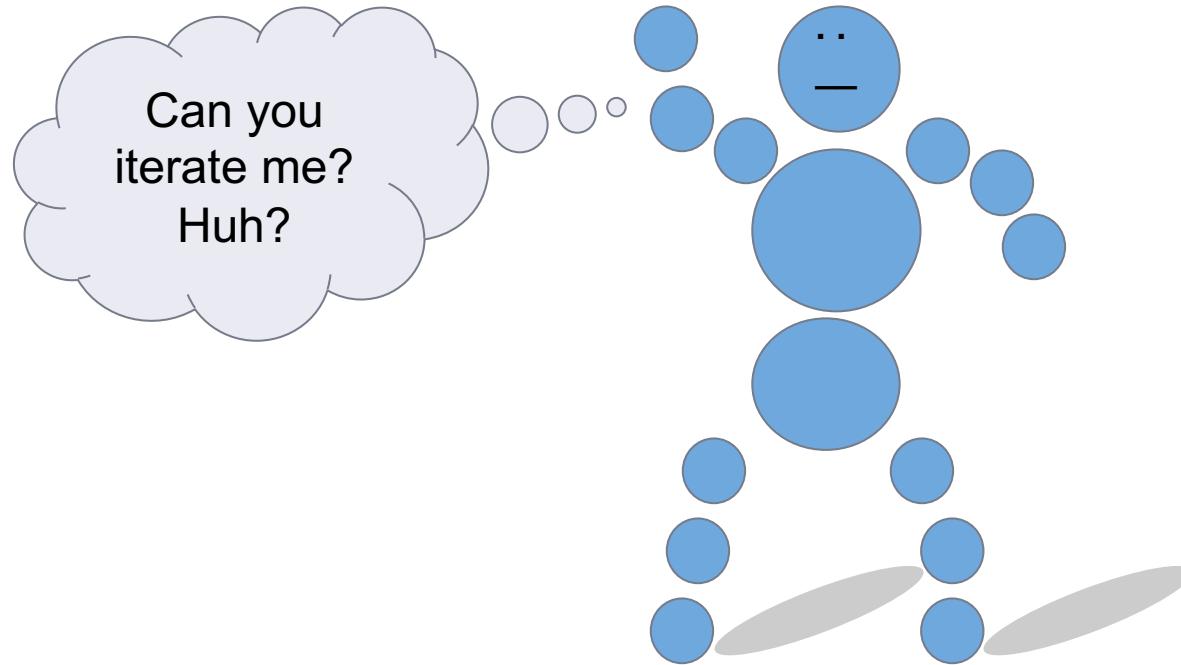
```
[1, 2, 3, 4, 5]
```



# Working with the Iterators

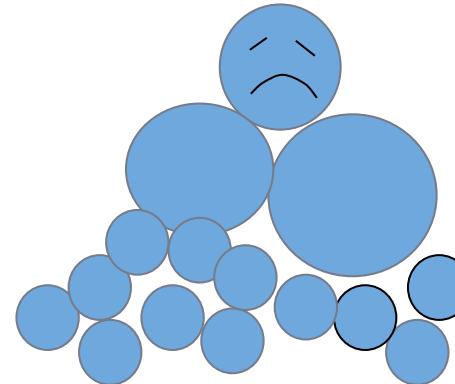
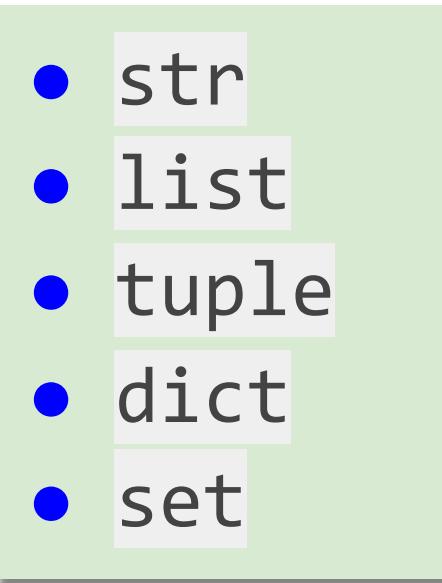


# Working with the Iterators



# Working with the Iterators (review)

- ▶ The most common iterable types :



# Working with the Iterators (review)

- ▶ Consider this example of a `str` type.

```
1 course = 'clarusway'  
2  
3 for i in course :  
4     print(i)  
5
```

# Working with the Iterators (review)

- ▶ Consider this example of a `str` type.

```
1 course = 'clarusway'  
2  
3 for i in course :  
4     print(i)  
5
```

```
1 c  
2 l  
3 a  
4 r  
5 u  
6 s  
7 w  
8 a  
9 y  
10
```

# Working with the Iterators

- ▶ **Task : Python Program to separate the string into its characters.**
  - ▶ Write a program to separate the string taken from the user into its characters using **for** loop.
  - ▶ Print the result such as :

```
input : "Clarusway"  
desired output : c-l-a-r-u-s-w-a-y
```

# Working with the Iterators

- ▶ The code might be like :

```
1 word = input("Give me a word :")
2 count = 0
3 for i in word:
4     count += 1
5     if count < len(word) :
6         i = i + "-"
7     print(i, end="")
8
9
```

input : "Clarusway"

Output

c-l-a-r-u-s-w-a-y

# Working with the Iterators

- ▶ Let's solve this without using a loop :

input : "Clarusway"

```
string = "clarusway"  
print("-".join(string))
```

Output

```
c-l-a-r-u-s-w-a-y
```

# Working with the Iterators

- ▶ Take a look at the other iterable type : **dict.**

```
1 user = {  
2     "name": "Daniel",  
3     "surname": "Smith",  
4     "age": 35  
5 }  
6  
7 for attribute in user:  
8     print(attribute)  
9 
```

What is the output? Try to figure out in your mind...



# Working with the Iterators

## ▶ The output :

```
1 user = {  
2     "name": "Daniel",  
3     "surname": "Smith",  
4     "age": 35  
5 }  
6  
7 for attribute in user:  
8     print(attribute)  
9 |
```



### Output

```
name  
surname  
age
```

# Working with the Iterators

- ▶ Take a look at the other iterable type : dict.

```
1 user = {  
2     "name": "Daniel",  
3     "surname": "Smith",  
4     "age": 35  
5 }  
6  
7 for i in user.values():  
8  
9     print (i, end=" ")  
10
```

What is the output? Try to figure out in your mind...

# Working with the Iterators

## ► The output :

```
1 user = {  
2     "name": "Daniel",  
3     "surname": "Smith",  
4     "age": 35  
5 }  
6  
7 for i in user.values():  
8  
9     print (i, end=" ")  
10
```

### Output

```
Daniel Smith 35
```

# Working with the Iterators

- ▶ Take a look at the other iterable type : dict.

```
1 user = {  
2     "name": "Daniel",  
3     "surname": "Smith",  
4     "age": 35  
5 }  
6  
7 for key, value in user.items():  
8     print (key,":",value)  
9
```

What is the output? Try to figure out in your mind...

# Working with the Iterators (review)

## ► The output :

```
1 user = {  
2     "name": "Daniel",  
3     "surname": "Smith",  
4     "age": 35  
5 }  
6  
7 for key, value in user.items():  
8     print (key, ":", value)  
9
```

## Output

```
name : Daniel  
surname : Smith  
age : 35
```



# Operations with `for` Loop



# Operations with the `for` Loop<sup>(review)</sup>

- In the following example, you'll get a **number** from the user and *print a sentence* the *number of times* we take from the user :

```
1 times = int(input("How many times should I say 'I love you'"))
2
3 for i in range(times):
4     print('I love you')
5
```

# Operations with the `for` Loop<sup>(review)</sup>

- In the following example, you'll get a number from the user and print a sentence the number of times we receive from the user :

```
1 times = int(input("How many times should I say 'I love you'"))
2
3 for i in range(times):
4     print('I love you')
```

Let's say the user enters **3**.

```
1 I love you
2 I love you
3 I love you
4
```

# Operations with the for Loop

- ▶ **Task :** This time, write a code block that asks the user a number between 1 and 10 then puts that number into the multiplication table.
- ▶ For example, the output for 5 should be as follows :

```
5x0 = 0
5x1 = 5
5x2 = 10
5x3 = 15
5x4 = 20
5x5 = 25
5x6 = 30
5x7 = 35
5x8 = 40
5x9 = 45
5x10 = 50
```

# Working with the Iterators

- ▶ The output can be like :

```
1 nmbr = int(input('enter a number between 1-10'))
2
3 for i in range(11):
4     print('{}x{} = '.format(nmbr, i), nmbr * i)
5
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ Let's take a close look at the `range()` function.
  - ▶ As we stated before, the formula syntax of the `range()` function is:

```
range(start, stop, step)
```

parameters

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
- ▶ Consider this example :

```
1 b = list(range(11))
2
3 print(b)
4
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ Let's take a close look at the `range()` function.
  - ▶ It creates an iterable sequence of numbers. And it can be simply converted into the `list`, `set`, and `tuple`.
  - ▶ Consider this example :

```
1 b = list(range(11))
2
3 print(b)
4
```

```
1 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Here's the other examples :

```
1 a = set(range(0,10))
2
3 print(a)
4
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Here's the other examples :

```
1 a = set(range(0,10))
2
3 print(a)
4
```

```
1 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
2
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Here's the other examples :

```
1 a = set(range(0,10))  
2  
3 print(a)  
4
```

```
1 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}  
2
```

```
1 c = tuple(range(11))  
2  
3 print(c)  
4
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Here's the other examples :

```
1 a = set(range(0,10))
2
3 print(a)
4
```

```
1 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
2
```

```
1 c = tuple(range(11))
2
3 print(c)
4
```

```
1 (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
2
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ An asterisk  \* separates the elements of the iterables.
- ▶ Let's take a look at an example of the `range()` function with starred  \* expression :

```
1 print(range(5)) # it will not print the numbers in sequence
2
3 print(*range(5)) # '*' separates its elements
4
```

What is the output? Try to figure out in your mind...

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Let's take a look at an example of the `range()` function with starred  \* expression :

```
1 print(range(5)) # it will not print the numbers in sequence
2
3 print(*range(5)) # '*' separates its elements
4
```

```
1 range(0, 5)
2 0 1 2 3 4
3
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Here's another example of the `range()` function with starred  \* expression :

```
1 print(*range(5,25,2))  
2
```

What is the output? Try to figure out in your mind...

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Here's another example of the `range()` function with starred  \* expression :

```
1 print(*range(5,25,2))  
2
```

```
1 5 7 9 11 13 15 17 19 21 23  
2
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Starred  \* expression can also be used to separate the other iterable objects. Such as str:

```
1 print(*('separate'))  
2
```

# Operations with the for Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ Starred  \* expression can also be used to separate the other iterable objects. Such as str:

```
1 print(*('separate'))  
2
```

```
1 s e p a r a t e  
2
```

# Operations with the `for` Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ You can create reverse sequence numbers using a negative **step**.

```
1 print(*range(10,0,-2))  
2
```

What is the output? Try to figure out in your mind...

# Operations with the `for` Loop<sup>(review)</sup>

- ▶ (... continued)
  - ▶ You can create reverse sequence numbers using a negative **step**.

```
1 print(*range(10,0,-2))  
2
```

```
1 10 8 6 4 2  
2
```

# Operations with the `for` Loop<sup>(review)</sup>

- ▶ Multiple variables in `for` loop.
  - ▶ Examine this example carefully :

`zip(iterator1, iterator2, ...)`

```
1 text = ['one', 'two', 'three', 'four', 'five']
2 numbers = [1, 2, 3, 4, 5]
3 for x, y in zip(text, numbers):
4     print(x, ':', y)
5
```

Use your IDEs

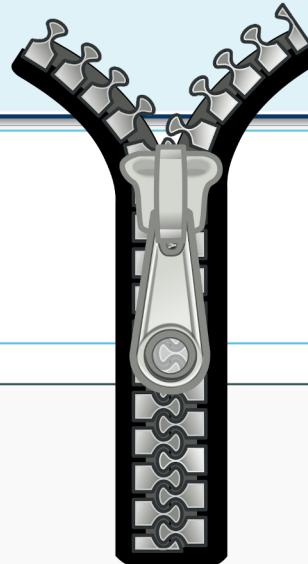
# Operations with the for Loop<sup>(review)</sup>

## Tips :

- `zip()` function make an iterator that aggregates elements from each of the iterables.

```
1 text = ['one', 'two', 'three', 'four', 'five']
2 numbers = [1, 2, 3, 4, 5]
3 for x, y in zip(text, numbers):
4     print(x, ':', y)
5
```

```
1 one : 1
2 two : 2
3 three : 3
4 four : 4
5 five : 5
6
```



# Operations with the for Loop

- ▶ **Task : Python Program to collect the odd and even numbers in two different lists.**
  - ▶ Write a program to choose and collect the even and odd numbers (1 to 10) in two different list.
  - ▶ Print the result such as :

```
evens: [0, 2, 4, 6, 8]
odds : [1, 3, 5, 7, 9]
```

# Operations with the for Loop

- ▶ The code might be like :

```
1 evens = []
2 odds = []
3
4 for n in range(10):
5     if n % 2 == 0:
6         evens.append(n)
7     else:
8         odds.append(n)
9
10 print(evens)
11 print(odds)
```

Output

```
[0, 2, 4, 6, 8]
[1, 3, 5, 7, 9]
```

# Operations with the for Loop

- ▶ **Task : Python Program to sum the amount of odd and even numbers in a tuple/list.**
  - ▶ Write a code that counts the odd and even numbers in a given **list** or **tuple**.
  - ▶ Print the result such as :

example list: [11, 2, 24, 61, 48, 33, 3]

example output : The number of even numbers : 3

The number of odd numbers : 4

# Operations with the for Loop

- ▶ The code might be like :

```
1 numbers = (11, 36, 33, 66, 89, 21, 32, 16, 10)
2 odds = 0
3 evens = 0
4 for i in numbers:
5     if not i % 2:
6         evens+=1
7     else:
8         odds+=1
9 print("The number of even numbers :", evens)
10 print("The number of odd numbers  :", odds)
11 |
```

## Output

```
The number of even numbers : 5
The number of odd numbers  : 4
```

# Operations with the `for` Loop

- ▶ **Task : Python Program to print out the numbers.**
  - ▶ Using the `for` loop, print the numbers from **1** to **9** as many as it is and get the following output.

```
1  
22  
333  
4444  
55555  
666666  
7777777  
88888888  
999999999
```

# Operations with the for Loop

- ▶ The code might be like :

```
1 | for i in range(1, 10):  
2 |     print(str(i) * i)  
3 |
```

# Operations with the `for` Loop

- ▶ **Task** : Python Program to sum of the numbers from **1** to **74**
  - ▶ Get the output of **2775** as a sum of the numbers between **1** - **74** (including).
  - ▶ Use `for` loop to make this calculation.

# Operations with the for Loop

- ▶ The code might be like :

```
1 sum_num=0
2
3 for i in range(1, 75):
4     sum_num += i
5
6 print(sum_num)
7
```



# Nested for Loop



# Nested for Loop (review)

- ▶ Simple structure of the nested **for** loops look like :

```
for variable1 in iterable1:  
    for variable2 in iterable2:  
        body
```

# Nested for Loop (review)

- ▶ Consider this example of the nested for loop :

```
1 who = ['I am ', 'You are ']
2 mood = ['happy', 'confident']
3 for i in who:
4     for ii in mood:
5         print(i + ii)
6
```

# Nested for Loop

- Consider this example of the nested for loop :

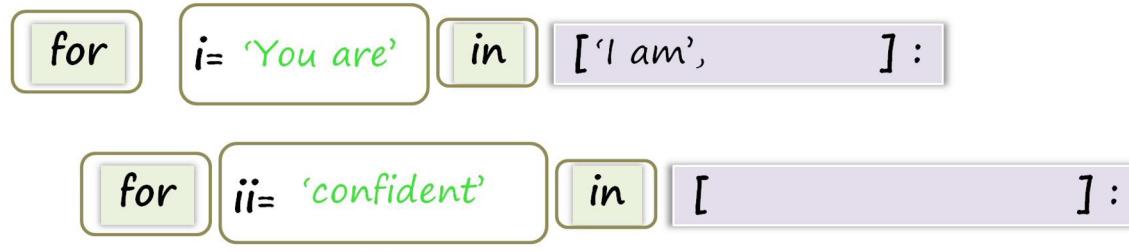
```
1 who = ['I am ', 'You are ']
2 mood = ['happy', 'confident']
3 for i in who:
4     for ii in mood:
5         print(i + ii)
6
```

First outer **then**  
inner loop runs.

```
1 I am happy
2 I am confident
3 You are happy
4 You are confident
5
```

# Nested for Loop (review)

- You can follow the animated diagram of this nested **for** loop for a better understanding.



```
print (i + ii)
```

*Output : I am happy  
I am confident  
You are happy*

# Nested for Loop

- ▶ **Task : Concatenation string elements from two separate lists.**
  - ▶ Write a code that takes string elements one by one and prints a sentence using nested **for** loops :
  - ▶ The given lists and sample outputs are :

```
names = ["susan", "tom", "edward"]  
mood = ["happy", "sad"]
```

example output : susan is happy  
                  susan is sad  
                  tom is happy

.

.



# Nested for Loop

- ▶ The code might be like :

```
1 names = ["susan", "tom", "edward"]
2 mood = ["happy", "sad"]
3
4 for i in names:
5     for ii in mood:
6         print(i + " is " + ii)
7 |
```

## Output

```
susan is happy
susan is sad
tom is happy
tom is sad
edward is happy
edward is sad
```



# List Comprehensions (List Generators)

Written in a long form:

```
liste = []  
for <var> in <iterable>:  
    <expression>
```

short form (Pythonic way of coding: Less code - more effectiveness.):

```
liste = [<expression> for <var> in <iterable>]
```



# List Comprehensions (List Generators)

Written in a long form:

```
squares = []
for i in range(10):
    squares.append(i * i)
```

The diagram illustrates the transformation of a long-form loop into a list comprehension. A blue curved arrow originates from the 'for' loop in the first code block and points to the 'for' part of the list comprehension in the second block. A green curved arrow originates from the 'squares.append(i \* i)' line in the first code block and points to the 'i \* i' part of the list comprehension's expression.

```
## short form
squares = [i * i for i in range(10)]
```



# List Comprehensions (List Generators)

Written in a long form;

```
for <var> in <iterable>:  
    if <condition>:
```

<expression>

short form;

```
[ <expression> for <var> in <iterable> if <condition> ]
```



# List Comprehensions (List Generators)

You create a list using a for loop and a **range()** function. (The following expression defines a generator for all the even numbers in 0-10):

```
evens = []
for n in range(12):
    if n % 2 == 0:
        evens.append(n)

print(evens)
```

Pythonic way of coding: Less code - more effectiveness.

```
evens = [n for n in range(12) if n%2 == 0]
print(evens)
```



# List Comprehensions (List Generators)

The following code stores words that contain the letter "a", in a list:

```
names_a = []
names = ['Python', 'Aisha', 'Bulend', 'Ala', 'Ahmed']

for word in names:
    if "a" in word.lower():
        names_a.append(word)

print(names_a)
```



# List Comprehensions (List Generators)

```
names_a = []
names = ['Python', 'Aisha', 'Bulend', 'Ala', 'Ahmed']

for word in names:
    if "a" in word.lower():
        names_a.append(word)

print(names_a)
```

This can be written in a single line, using a list comprehension:

```
names = ['Python', 'Aisha', 'Bulend', 'Ala', 'Ahmed']
names_a = [word for word in names if "a" in word.lower()]
print(names_a)
```



# List Comprehensions (List Generators)

Written in a long form:

```
for <var> in <iterable>:  
    if <condition>:
```

```
        <expression-1>  
    else:  
        <expression-2>
```

```
[<expr1> if condition else <expr2> for <var> in iterable]
```

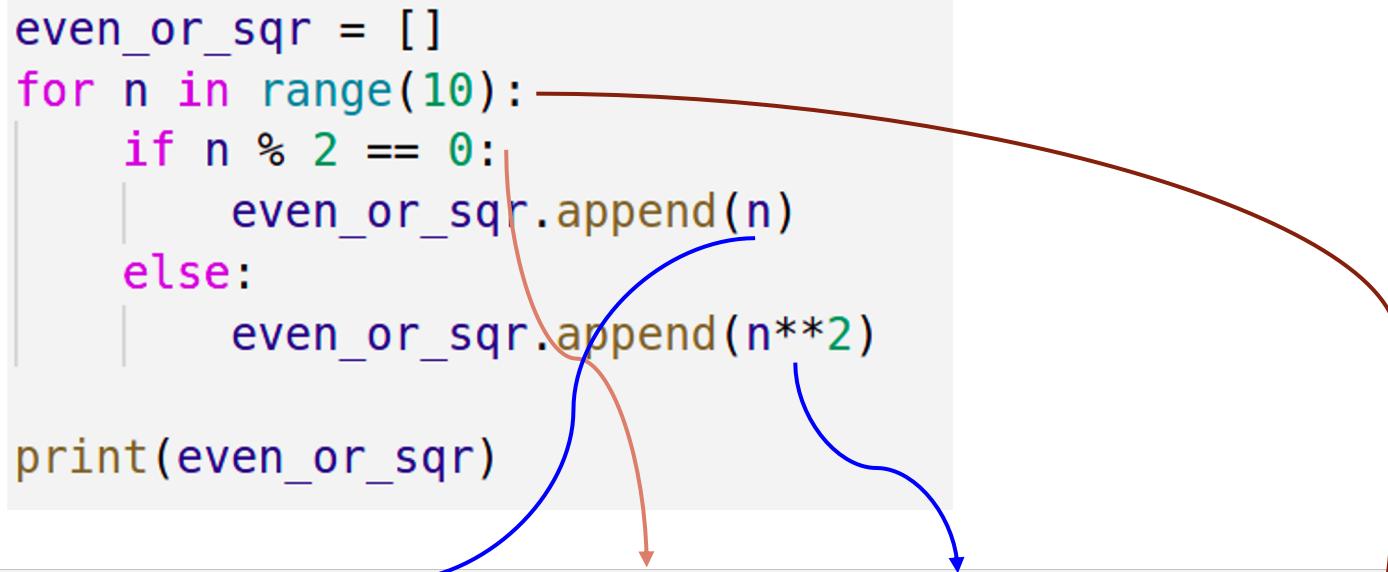


# List Comprehensions (List Generators)

Written in a long form;

```
even_or_sqr = []
for n in range(10):
    if n % 2 == 0:
        even_or_sqr.append(n)
    else:
        even_or_sqr.append(n**2)

print(even_or_sqr)
```



short form;

```
even_or_sqr = [n if n % 2 == 0 else n**2 for n in range(10)]
print(even_or_sqr)
```

# List Comprehensions (List Generators)



## #list comprehension samples:

```
[i**2 for i in range(10)]      #out-1: ?
```

```
sum([n for n in range(75)])      #out-2: ?
```

```
cumle = "Mistakes are for people"
```

```
set(i for i in cumle if not i in 'aeiou')  #out-3: ?
```