

# Programming Thinking

## Session 5

Pepe García [jgarciah@faculty.ie.edu](mailto:jgarciah@faculty.ie.edu)



SCHOOL OF  
SCIENCE &  
TECHNOLOGY

# Plan for this session

- lists

# Plan for this session

- lists
- iteration

# Mutability refresher

Mutability is a feature of variables in most programming languages. It means that variables can be updated to newer values.

```
x = 1
x = x * 3

print(x)
```

# Mutability refresher

Mutability is a feature of variables in most programming languages. It means that variables can be updated to newer values.

```
x = 1
x = x * 3

print(x)
```

## Demo

Let's refresh how we can change the value of a variable in Spyder.

Iteration is the act of repeating a process. In Python we express iteration with the **while** statement

Iteration is the act of repeating a process. In Python we express iteration with the **while** statement

```
while <condition>:  
    <body>
```

Iteration is the act of repeating a process. In Python we express iteration with the **while** statement

```
while <condition>:  
    <body>
```

- 1 Evaluate the condition



Iteration is the act of repeating a process. In Python we express iteration with the **while** statement

```
while <condition>:  
    <body>
```

- 1 Evaluate the condition
- 2 If the condition is False, exit while and go to next statement

Iteration is the act of repeating a process. In Python we express iteration with the **while** statement

```
while <condition>:  
    <body>
```

- 1 Evaluate the condition
- 2 If the condition is False, exit while and go to next statement
- 3 If condition is true, execute body. Then go to step 1.

## Demo

Using iteration, let's print integers from zero to 50



## Exercise 1

Create a function `pyramid` that receives an integer `n` as parameter and prints `n` lines of the following pattern:

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*



# Checkpoint

Are there questions so far?



SCHOOL OF  
SCIENCE &  
TECHNOLOGY

Lists are a sequence data structure. We can store multiple values inside them, add and remove elements, update them, concatenate them, etc.

# Constructing Lists

We construct lists with the brackets `[]` syntax. We surround everything that we want to include in the list with **square brackets** and separate elements with **commas**:

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

```
["hello", "dolly"]
```

```
[]
```

```
[1, "hello", 2, "dolly", 3]
```

# Constructing Lists

We construct lists with the brackets `[]` syntax. We surround everything that we want to include in the list with **square brackets** and separate elements with **commas**:

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

```
["hello", "dolly"]
```

```
[]
```

```
[1, "hello", 2, "dolly", 3]
```

## Demo

Let's create some lists and see the type of those values.





# List length

We can get the length of a list using the **len** function:

```
names = ["Pepe", "Antonio"]  
print(len(names)) # will print 2
```



# Accessing list elements

We use **square brackets** to access elements by their **index**.

## Indices

**indices** in lists start by **0**, not 1.

```
words = ["hello", "dolly"]  
print(words[0])  
# prints "hello"  
print(words[1])  
# prints "dolly"
```



# Accessing list elements

We use **square brackets** to access elements by their **index**.

## Indices

**indices** in lists start by **0**, not 1.

```
words = ["hello", "dolly"]  
print(words[0])  
# prints "hello"  
print(words[1])  
# prints "dolly"
```

## Demo



## Exercise 2

Create a function that receives a list as a parameter and prints each element of the list individually.

# Operators on lists

As with strings, `+` and `*` operators work with lists too!

# Operators on lists

As with strings, `+` and `*` operators work with lists too!

Demo

# Mutating lists

Lists are mutable values, and they provide functionality to add, delete, and update elements

# Updating elements in the list

To update an element inside the list, we use a syntax similar to the one for declaring variables, but using the brackets and the index we refer to.

```
numbers = [1,2,4]
numbers[2] = 3
print(numbers) # prints [1,2,3]
```





# Updating elements in the list

To update an element inside the list, we use a syntax similar to the one for declaring variables, but using the brackets and the index we refer to.

```
numbers = [1,2,4]
numbers[2] = 3
print(numbers) # prints [1,2,3]
```

## Demo

# Appending elements to the list

To add a new element to the end of the list we use the `append()` method on it.

```
numbers = [1,2,3]
numbers.append(4)
print(numbers) # prints [1,2,3,4]
```

# Appending elements to the list

To add a new element to the end of the list we use the `append()` method on it.

```
numbers = [1,2,3]
numbers.append(4)
print(numbers) # prints [1,2,3,4]
```

## Demo



# Inserting elements in the list

There's an alternative way of adding new elements to the list, and it's using the `insert()` method on it:

```
words = ["hello", "my", "friends"]  
words.insert(2, "dear")  
print(words) # prints ["hello", "my", "dear", "friends"]
```

The difference between this and `append` is that with `insert` we can choose where to put it by using the target index

# Inserting elements in the list

There's an alternative way of adding new elements to the list, and it's using the `insert()` method on it:

```
words = ["hello", "my", "friends"]  
words.insert(2, "dear")  
print(words) # prints ["hello", "my", "dear", "friends"]
```

The difference between this and `append` is that with `insert` we can choose where to put it by using the target index

## Demo

# Removing elements from the list

In order to remove an element from a list, we should use the `.pop()` method, and pass the index of the element we want to remove

```
words = ["hello", "my", "friend"]  
words.pop(1)  
print(words)
```

# Removing elements from the list

In order to remove an element from a list, we should use the `.pop()` method, and pass the index of the element we want to remove

```
words = ["hello", "my", "friend"]  
words.pop(1)  
print(words)
```

## Demo



# For loops

For loops are simpler than while loops. They iterate over elements in a list. They loop once for every element in the list.





# For loops

For loops are simpler than while loops. They iterate over elements in a list. They loop once for every element in the list.

```
for <iteration_variable> in <list>:  
    <body>
```

In each iteration, we will have a new value for the `iteration_variable`.

# For loops

For loops are simpler than while loops. They iterate over elements in a list. They loop once for every element in the list.

```
for <iteration_variable> in <list>:  
    <body>
```

In each iteration, we will have a new value for the `iteration_variable`.

## Demo

## Exercise

create a function named **to\_string** that receives a list and returns a string with all elements of the list concatenated. Don't use the `join` function.

# Recap

We will use **while** loops for iterating given a boolean condition.



We will use **while** loops for iterating given a boolean condition.

Use **lists** to store collections of values



We will use **while** loops for iterating given a boolean condition.

Use **lists** to store collections of values

Use mutation operations on list to append, remove, or update elements in the list

We will use **while** loops for iterating given a boolean condition.

Use **lists** to store collections of values

Use mutation operations on list to append, remove, or update elements in the list

Use for loops to iterate over elements in the list



- Create a function that returns a list of numbers from 0 to 500
- Create a function that takes a list of numbers (you can use the one you created in the previous exercise) and returns the sum of all of them
- Investigate the `range()` function. After you've used it, create a function that receives a number as parameter and prints all numbers from it to zero (using a for loop).
- Create a function that takes a list of numbers and returns the maximum value among them
- Create a function that takes a list of numbers and returns the minimum value among them

(cont)

# Exercises (Cont)

- Create a function that prints the numbers 1 to 50 (using iteration)
- Create a program that prints multiplication tables from 1 to 10
- Create a function `inverted_piramid` that writes the pyramid of stars in an inverted fashion.

\*\*\*\*\*

\*\*\*\*

\*\*\*

\*\*

\*

(cont)

# Exercises (Cont)

- Create a function `multiply` that takes two integers (`a` and `b`, for example) and returns `a` times `b`. Do not use the `*` operator.
- Create a function `exponentiate` that takes two arguments `base` and `exponent` and raises `base` to the `exponent` power. Do not use the `**` operator.

# Recommended literature

<https://www.py4e.com/html3/05-iterations>

<https://www.py4e.com/lessons/lists>



SCHOOL OF  
SCIENCE &  
TECHNOLOGY