MY JOURNAL TO PYTHON

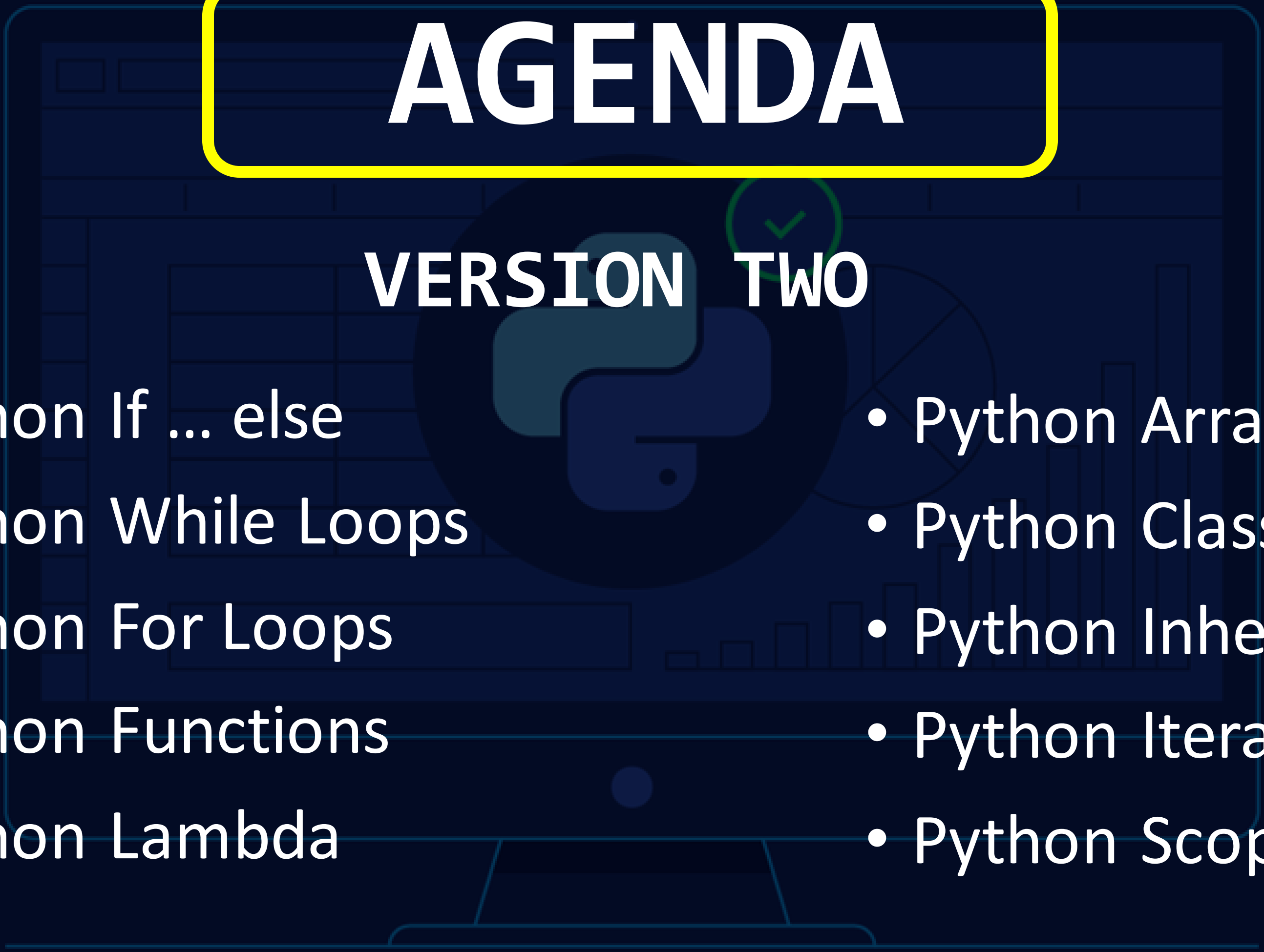
NAME: 喬萬斯

Teacher: My dear great teacher

4110E234

AGENDA

VERSION TWO



- Python If … else
- Python While Loops
- Python For Loops
- Python Functions
- Python Lambda

- Python Arrays
- Python Classes Objects
- Python Inheritance
- Python Iterators
- Python Scope

Python If ... Else

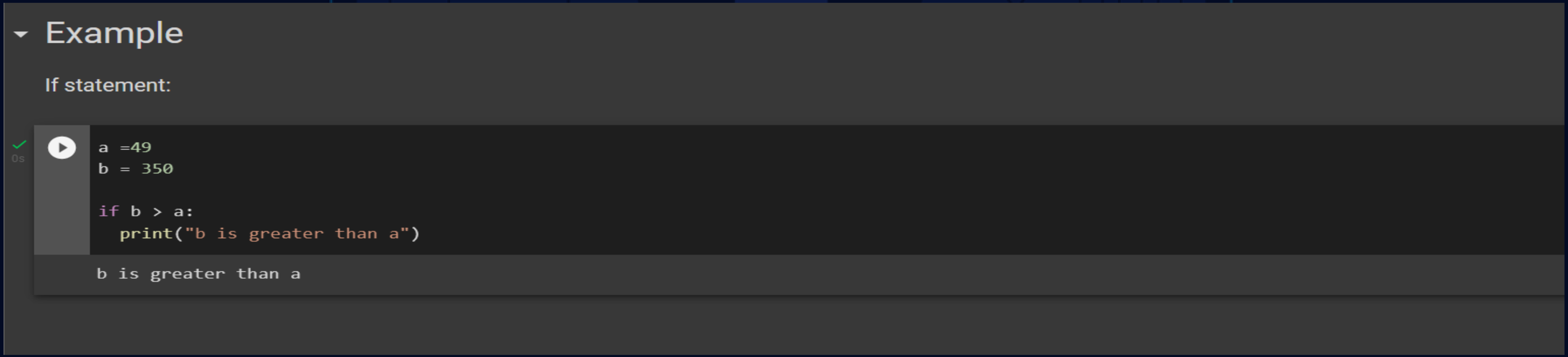
Python Conditions and If statements

Python supports the standard mathematical logical conditions:

- Equals: a == b
- Not Equals: a != b
- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b

Python If ... Else

These conditions can be utilized in a variety of ways, the most frequent of which being "if statements" and loops.
The if keyword is used to create a "if statement."



If statement:

```
a =49
b = 350

if b > a:
  print("b is greater than a")
```

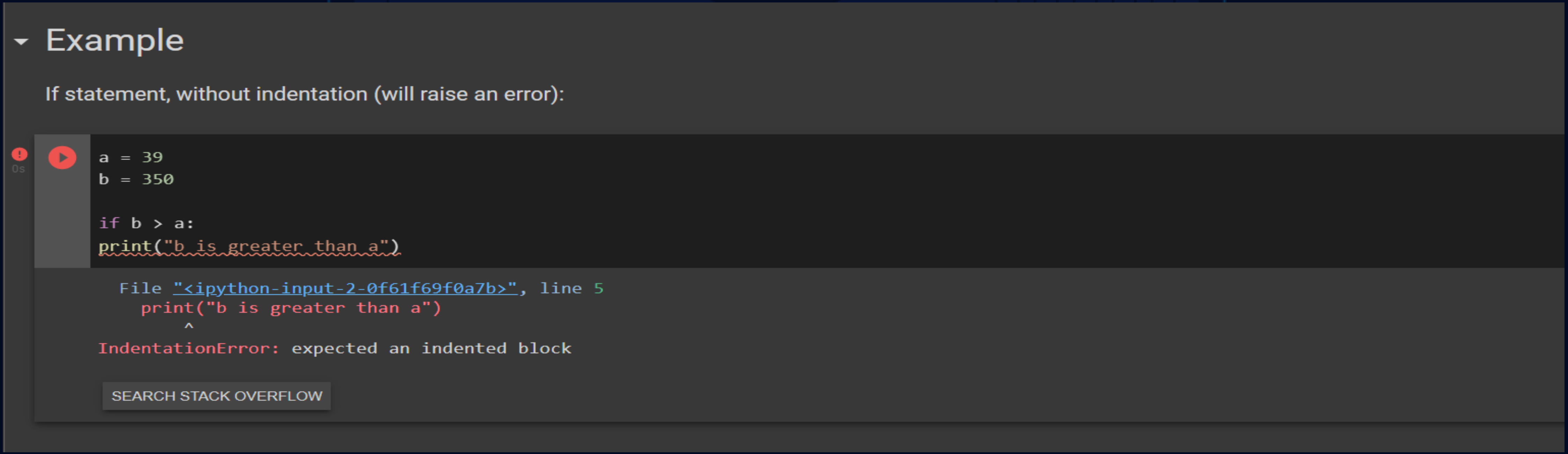
```
b is greater than a
```

In this example, two variables, a and b, are used in the if statement to determine whether b is greater than a. Because an is 49 and b is 350, we know that 350 is greater than 49, thus we print "b is greater than a" on the screen.

Python If ... Else

Indentation

Python uses indentation (whitespace at the start of a line) to define scope in code. Curly-brackets are frequently used for this purpose in other programming languages.



Example

If statement, without indentation (will raise an error):

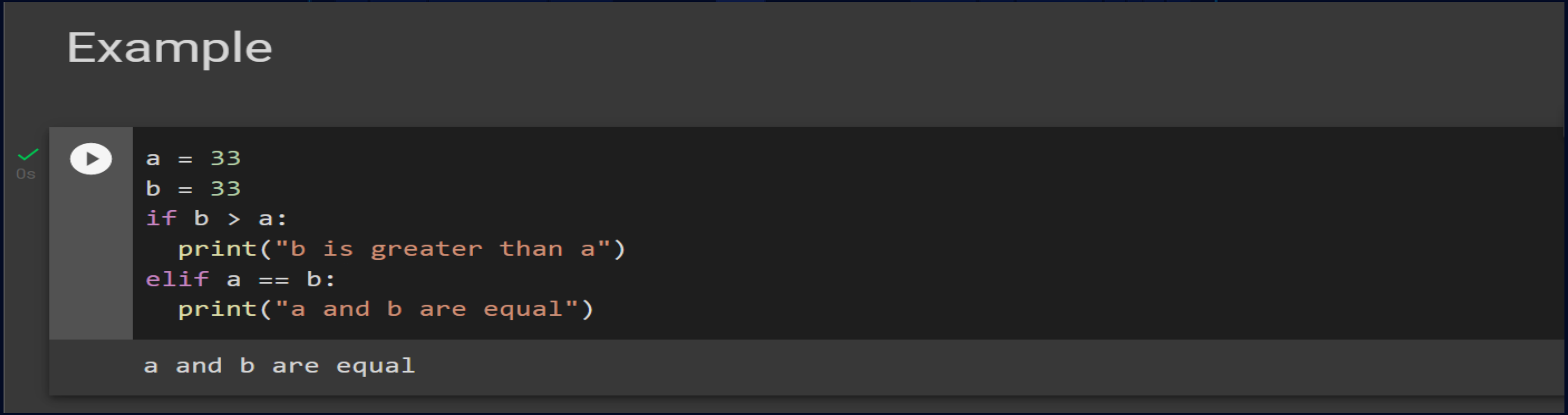
```
a = 39
b = 350

if b > a:
print("b is greater than a")
```

```
  File "<ipython-input-2-0f61f69f0a7b>", line 5
    print("b is greater than a")
        ^
IndentationError: expected an indented block
```

Python If ... Else

Elif

The elif keyword in Python means "attempt this condition if the previous conditions were not true."



Example

```
a = 33
b = 33
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
```

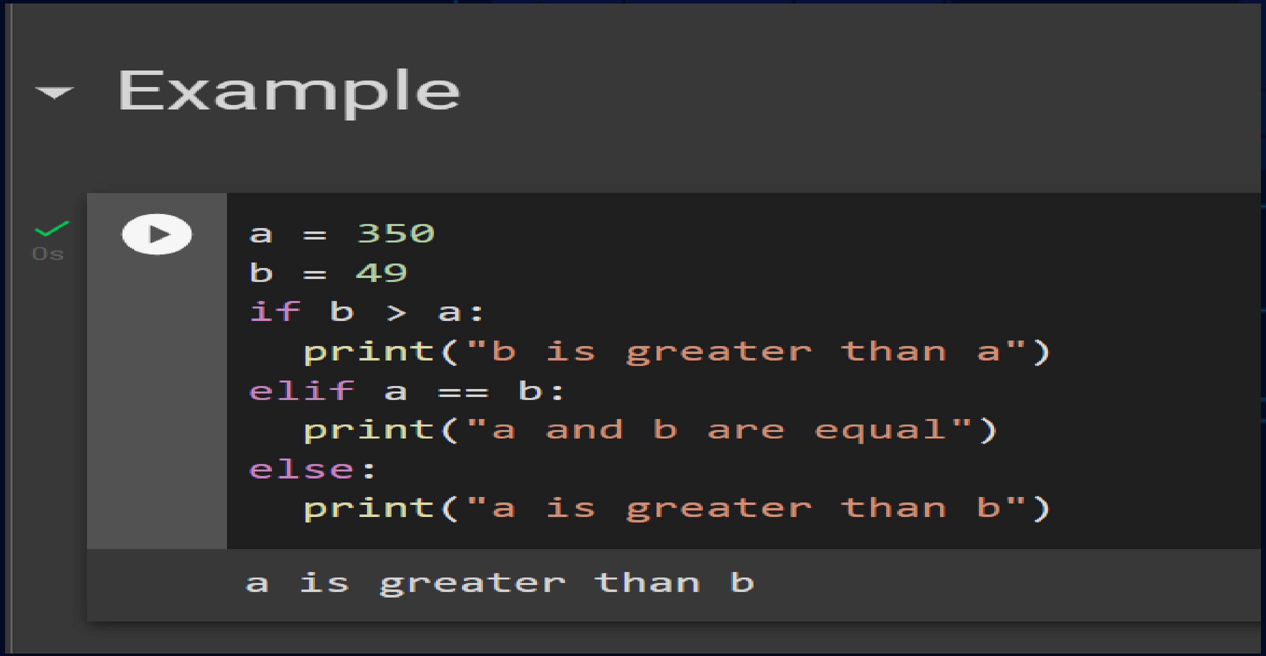
```
a and b are equal
```

In this example, a equals b, so the first condition is false, but the elif condition is true, so we print "a and b are equal" to the screen.

Python If ... Else

Else

Because an equals b in this example, the first condition is false, but the elif condition is true, and we output "a and b are equal" on the screen.



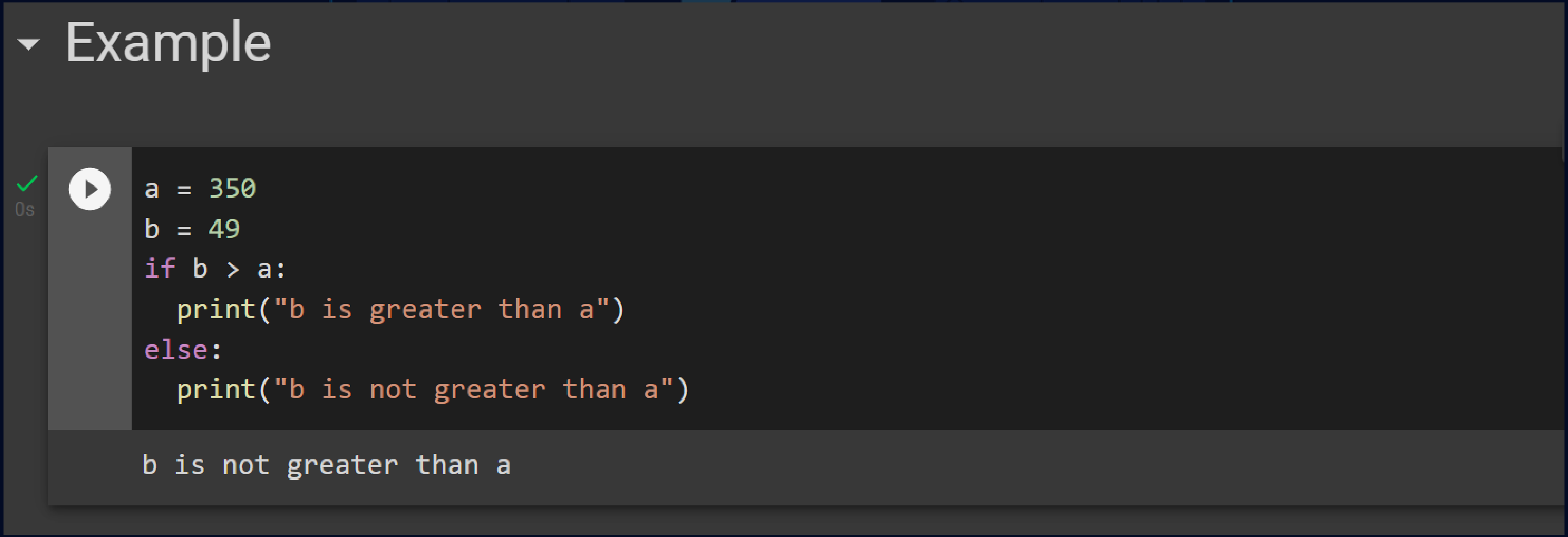
```
a = 350
b = 49
if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")
```

```
a is greater than b
```

In this example, a is greater than b, so the first condition is false, as is the elif condition, so we go to the else condition and print "a is greater than b" to the screen.

Python If ... Else

You can also use else instead of elif:



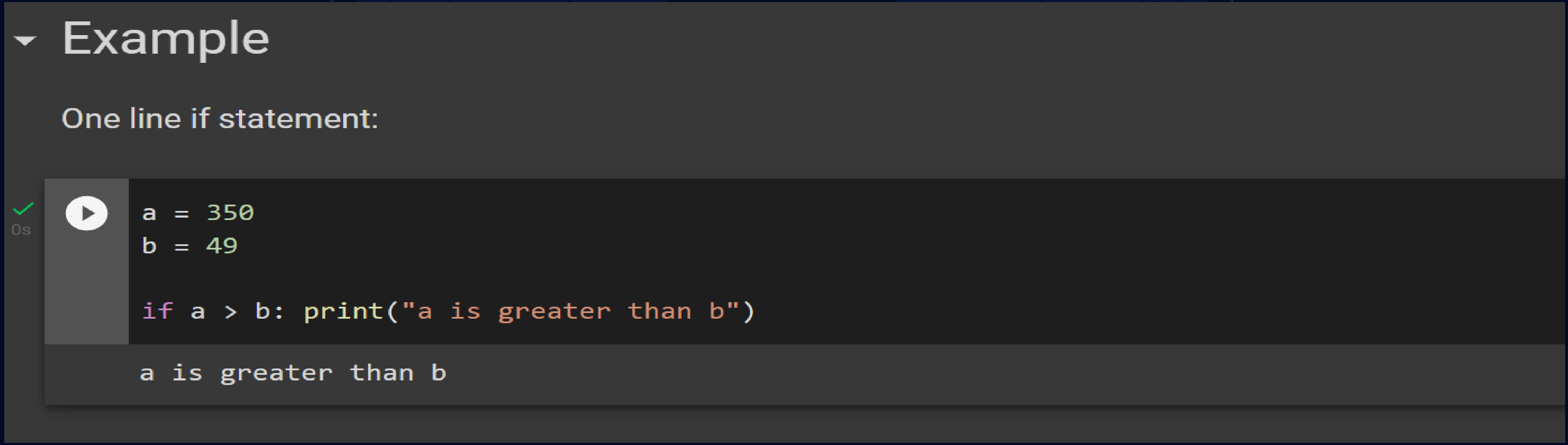
```python
a = 350
b = 49
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

```
b is not greater than a
```

Python If ... Else

Short Hand If

If you only need to execute one statement, one for if and one for else, you can put it all on the same line:



Example

One line if statement:

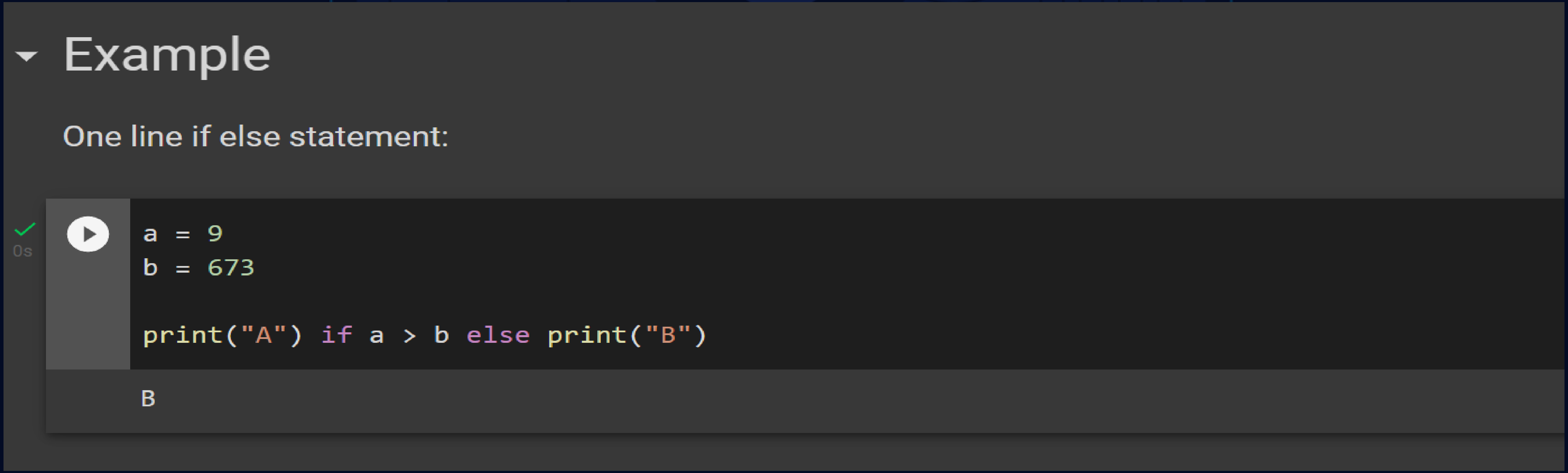
```
a = 350
b = 49

if a > b: print("a is greater than b")
```

```
a is greater than b
```

Python If ... Else

Short Hand If ... Else

If you have only one statement to execute, one for if, and one for else, you can put it all on the same line:



Example

One line if else statement:

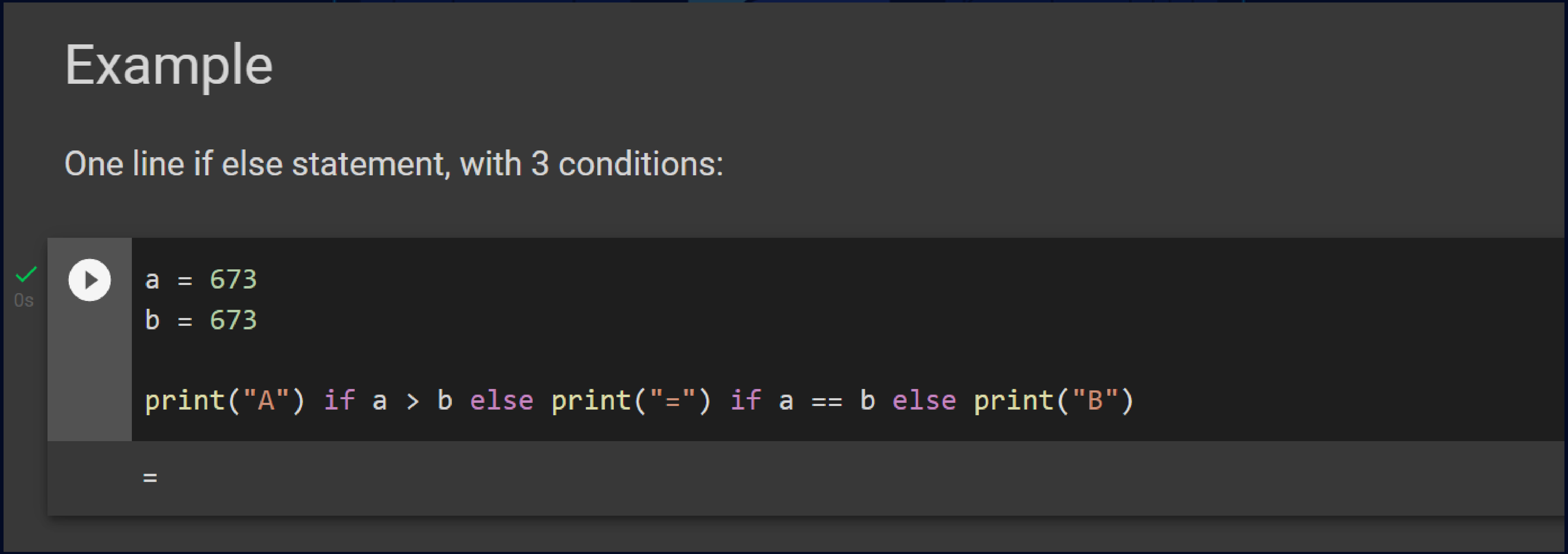
```
a = 9
b = 673

print("A") if a > b else print("B")
```

```
B
```

This is referred to as Ternary Operators or Conditional Expressions.

Python If ... Else

On the same line, you can also have numerous else statements:



Example

One line if else statement, with 3 conditions:

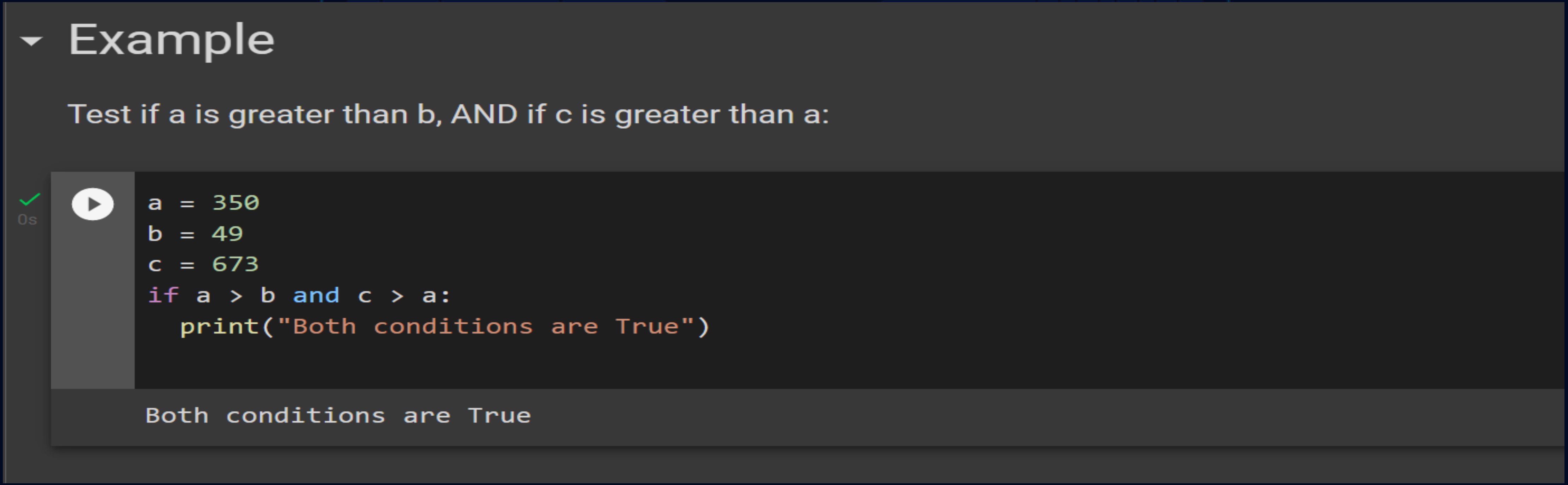
```
a = 673
b = 673

print("A") if a > b else print("=") if a == b else print("B")
```

```
=
```

Python If ... Else

And

The And keyword is a logical operator for combining conditional statements:



Example

Test if a is greater than b, AND if c is greater than a:

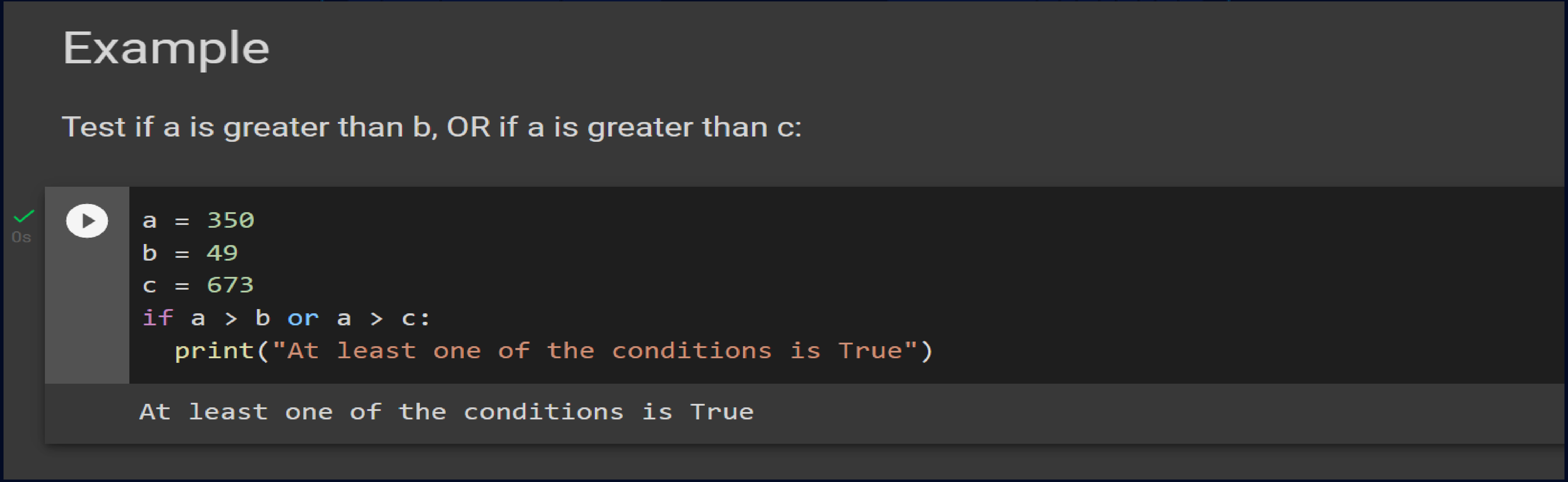
```python
a = 350
b = 49
c = 673
if a > b and c > a:
  print("Both conditions are True")
```

```
Both conditions are True
```

Python If ... Else

Or

The or keyword is a logical operator for combining conditional statements:



Example

Test if a is greater than b, OR if a is greater than c:

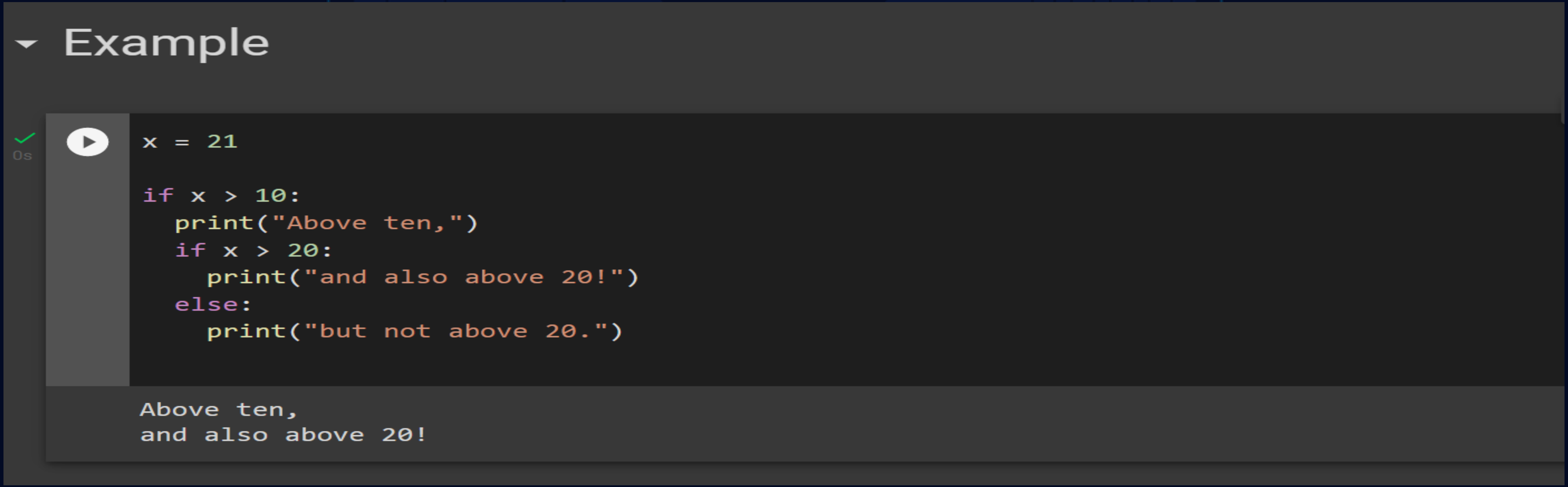
```
a = 350
b = 49
c = 673
if a > b or a > c:
  print("At least one of the conditions is True")
```

```
At least one of the conditions is True
```

Python If ... Else

Nested If

If statements can be nestled inside other if statements, which is known as nested if statements.



Example

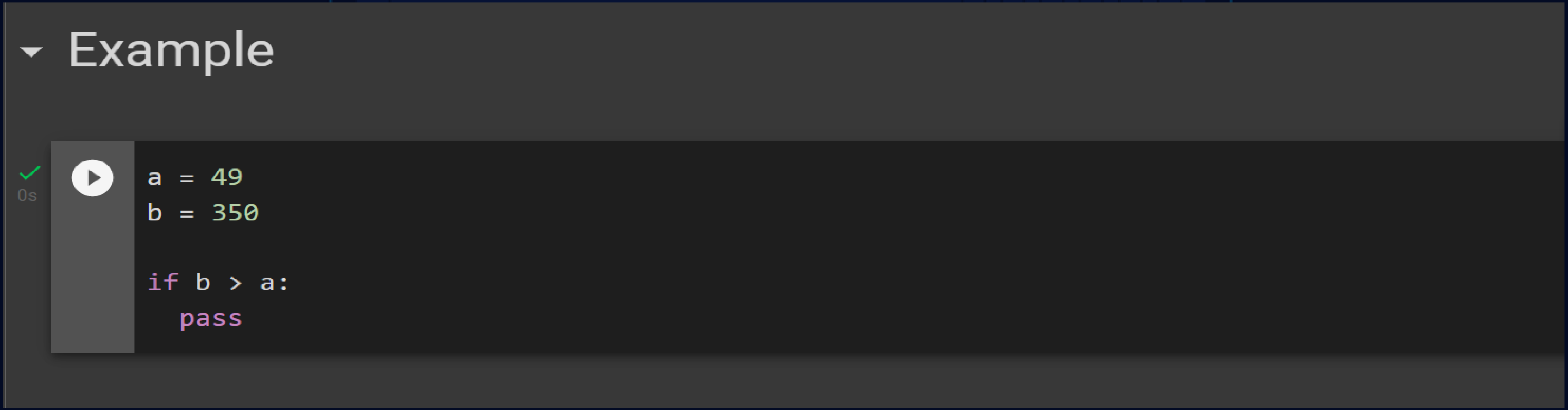
```
x = 21

if x > 10:
  print("Above ten,")
  if x > 20:
    print("and also above 20!")
  else:
    print("but not above 20.")
```

```
Above ten,
and also above 20!
```

Python If ... Else

The pass Statement

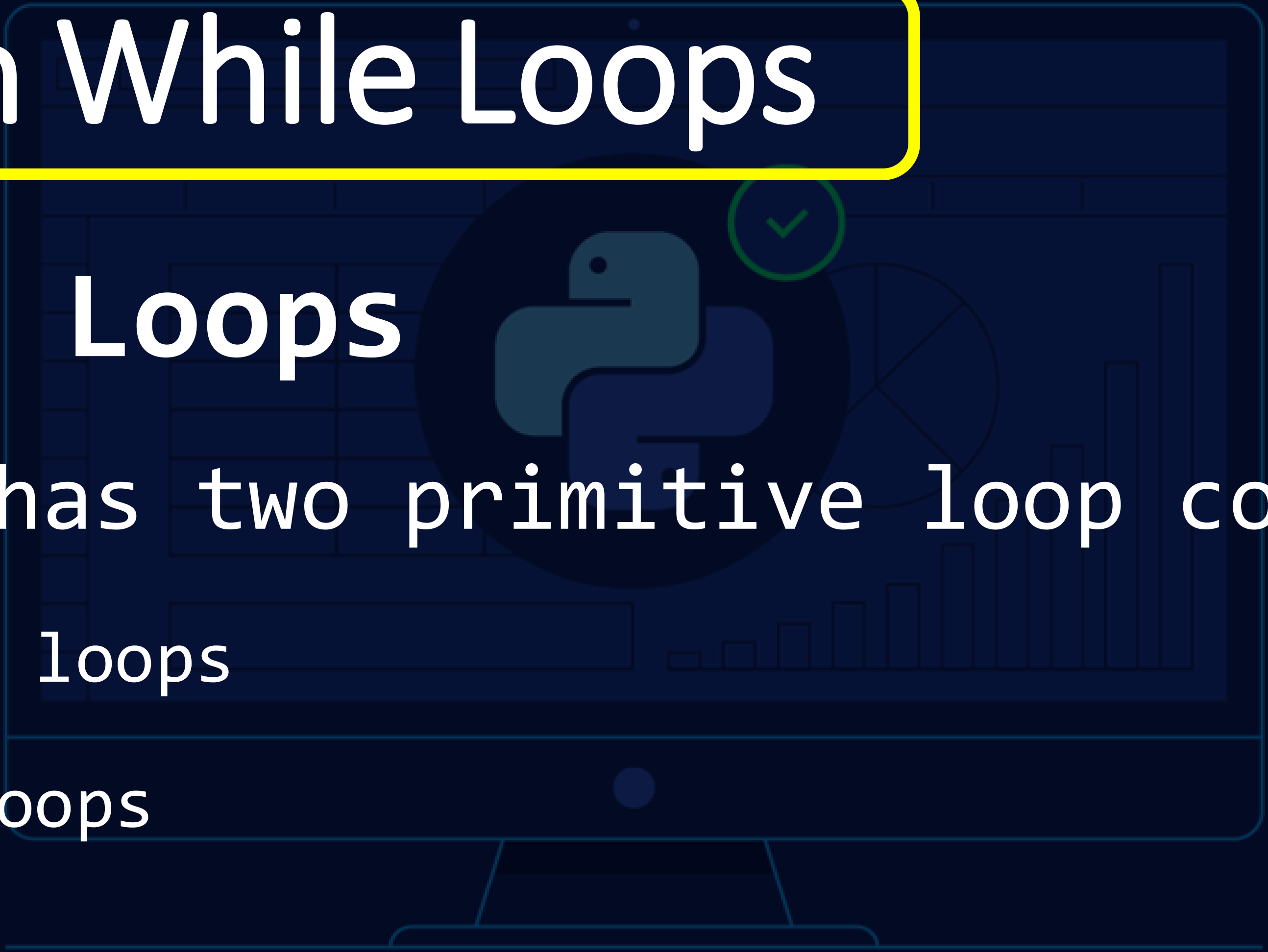
If statements cannot be empty, but if you have an if statement with no content, include the pass statement to prevent an error.



Example

```
a = 49
b = 350

if b > a:
  pass
```

Python While Loops



Python Loops

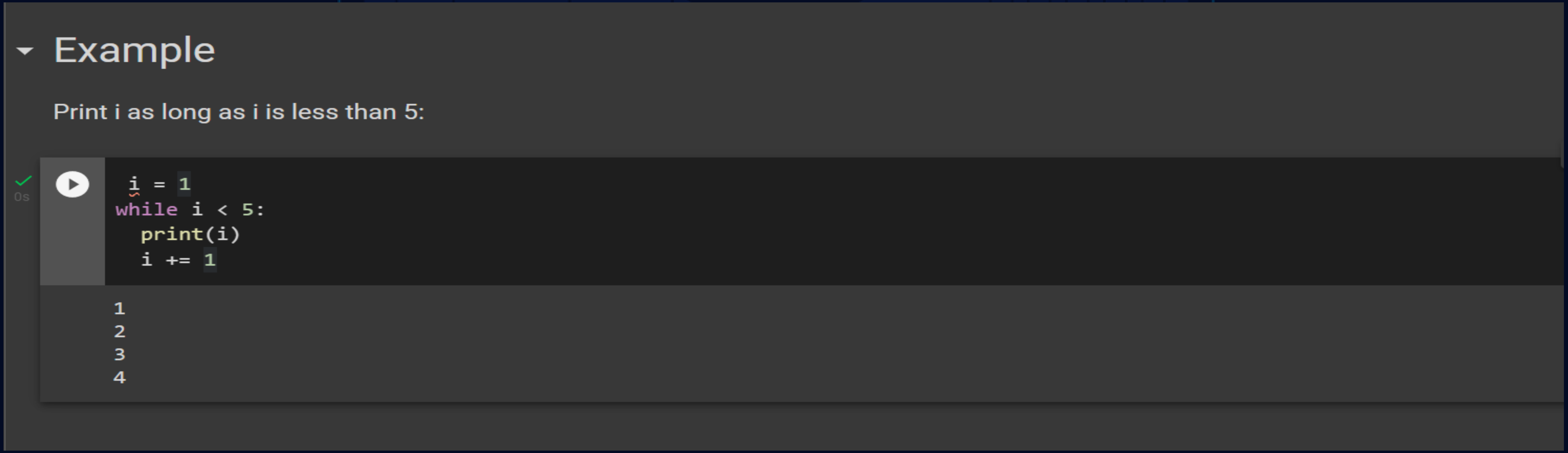
Python has two primitive loop commands:

- `while` loops
- `for` loops

Python While Loops

The while Loop

We can use the while loop to execute a series of statements as long as a condition is true.



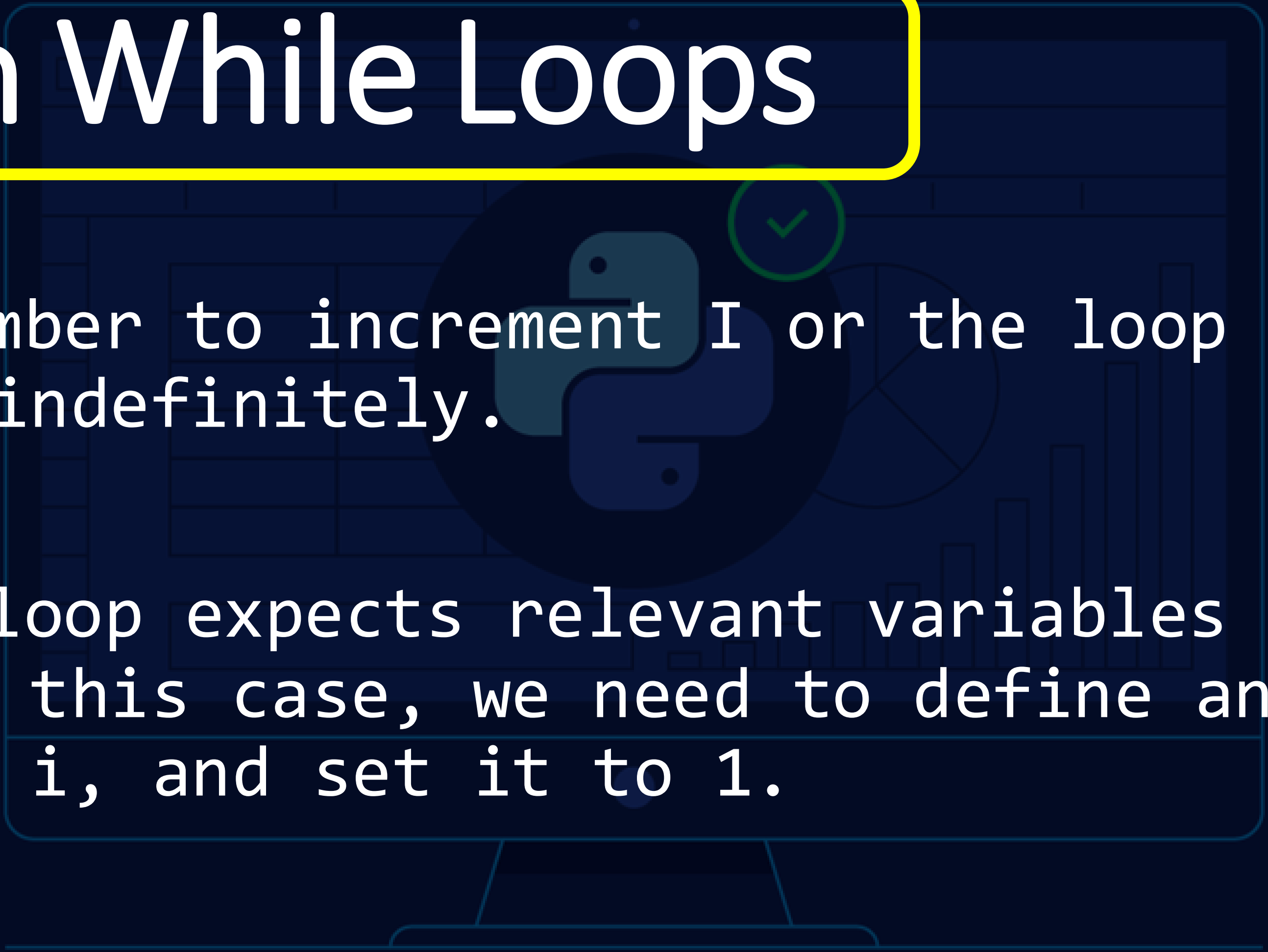
Example

Print i as long as i is less than 5:

```
i = 1
while i < 5:
  print(i)
  i += 1
```

```
1
2
3
4
```

Python While Loops



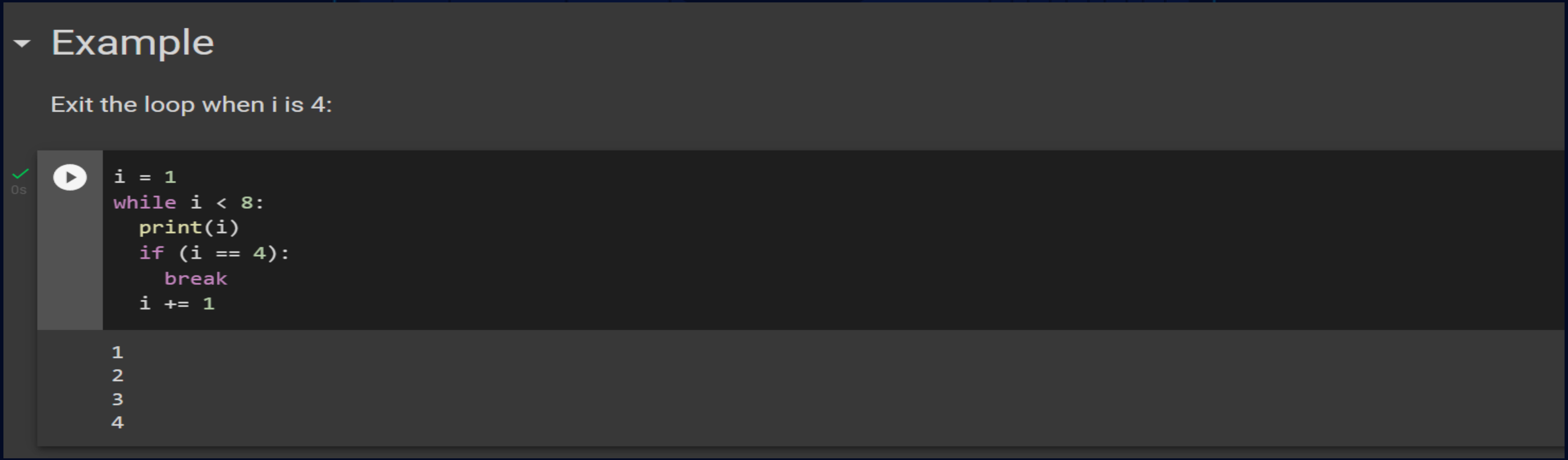
Note: Remember to increment I or the loop will continue indefinitely.

The while loop expects relevant variables to be ready; in this case, we need to define an indexing variable, i, and set it to 1.

Python While Loops

The break Statement

We can break the loop even if the while condition is true by using the break statement:



Example

Exit the loop when i is 4:

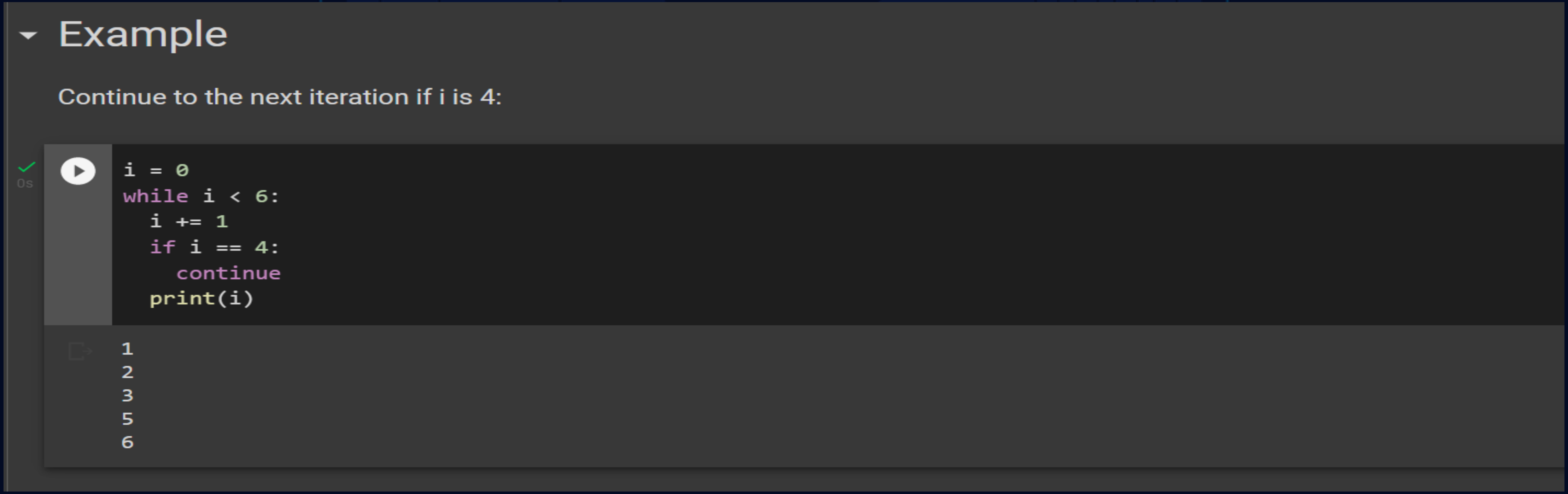
```python
i = 1
while i < 8:
  print(i)
  if (i == 4):
    break
  i += 1
```

```
1
2
3
4
```

Python While Loops

The continue Statement

We can use the continue statement to end the current iteration and begin the next:



Example

Continue to the next iteration if i is 4:

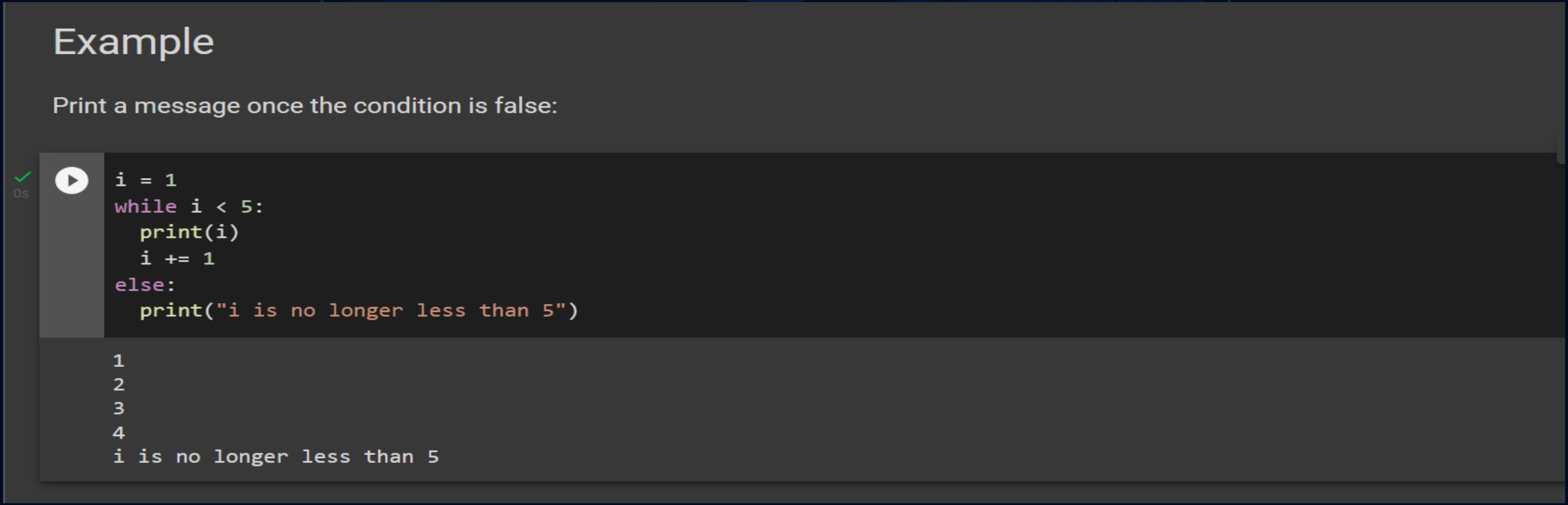
```
i = 0
while i < 6:
  i += 1
  if i == 4:
    continue
  print(i)
```

```
1
2
3
5
6
```

Python While Loops

The else Statement

We can use the else statement to run a block of code only once when the condition is no longer true:



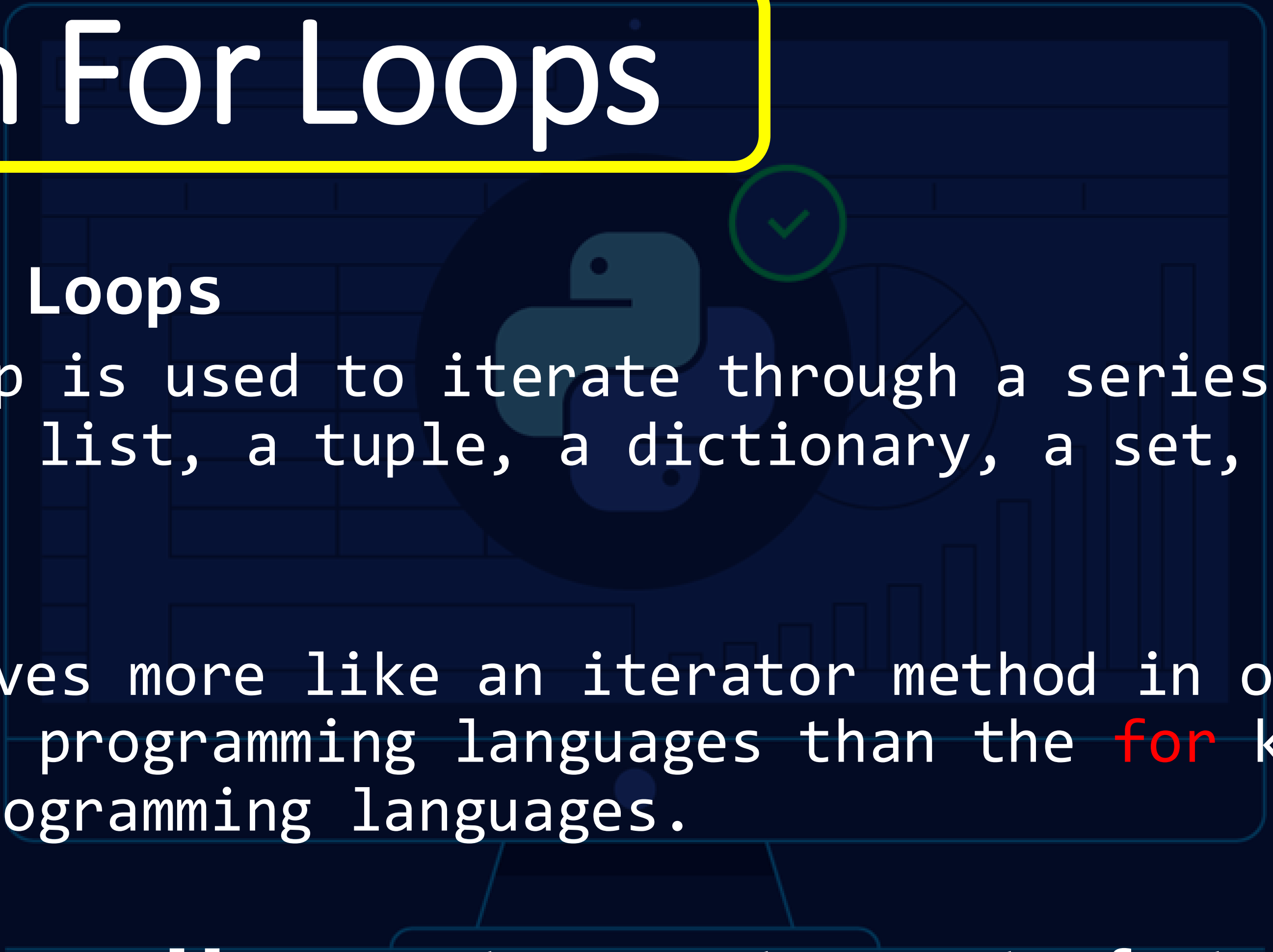
Example

Print a message once the condition is false:

```
i = 1
while i < 5:
  print(i)
  i += 1
else:
  print("i is no longer less than 5")
```

```
1
2
3
4
i is no longer less than 5
```

Python For Loops



Python For Loops

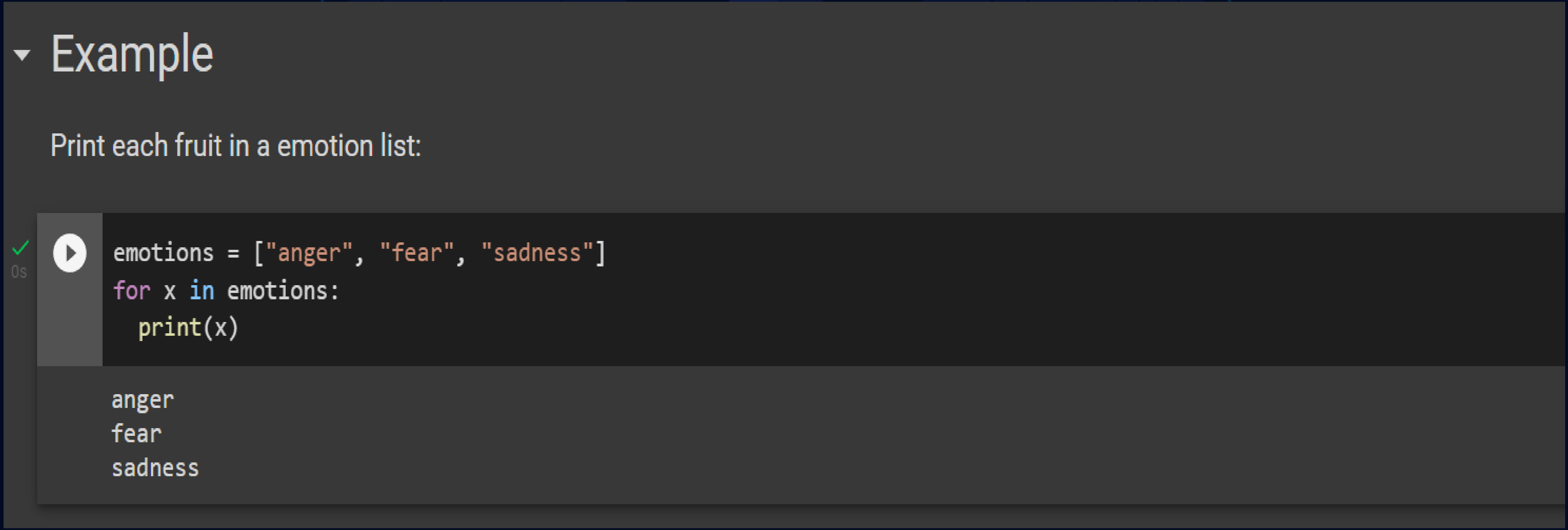
A for loop is used to iterate through a series (that is either a list, a tuple, a dictionary, a set, or a string).

This behaves more like an iterator method in other object-oriented programming languages than the for keyword in other programming languages.

The for loop allows us to execute a set of statements once for each item in a list, tuple, set, and so on.

Python For Loops

There is no need to set an indexing variable before using the for loop.



Example

Print each fruit in a emotion list:

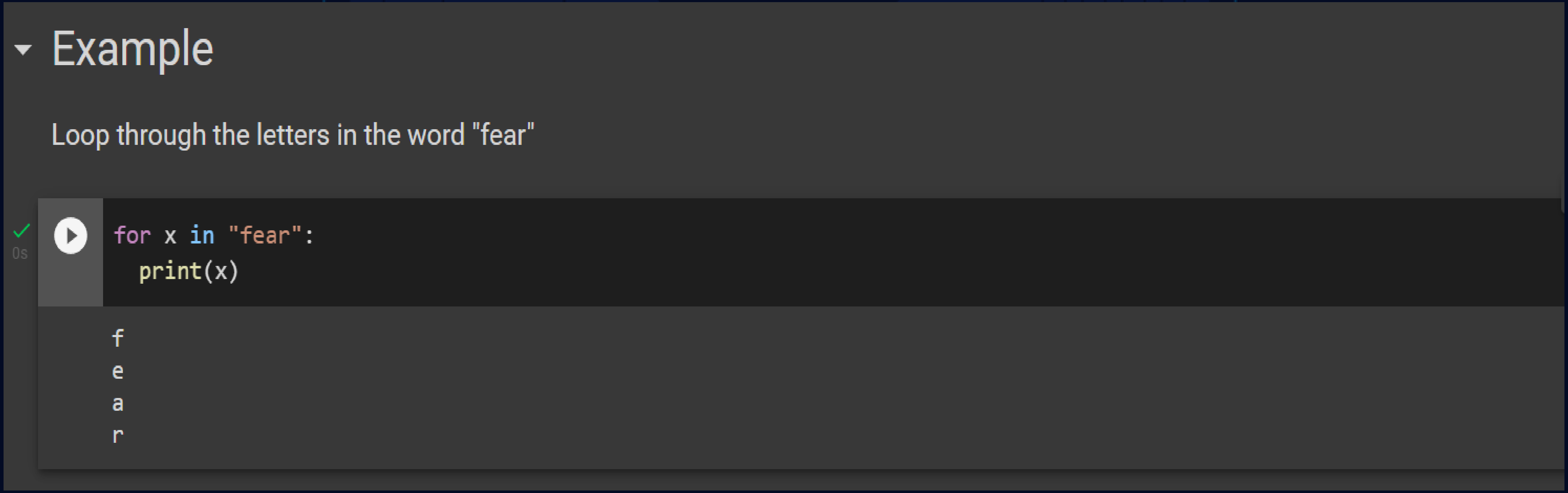
```
emotions = ["anger", "fear", "sadness"]
for x in emotions:
  print(x)
```

```
anger
fear
sadness
```

Python For Loops

Looping Through a String

Strings are also iterable objects because they contain a sequence of characters:



Example

Loop through the letters in the word "fear"

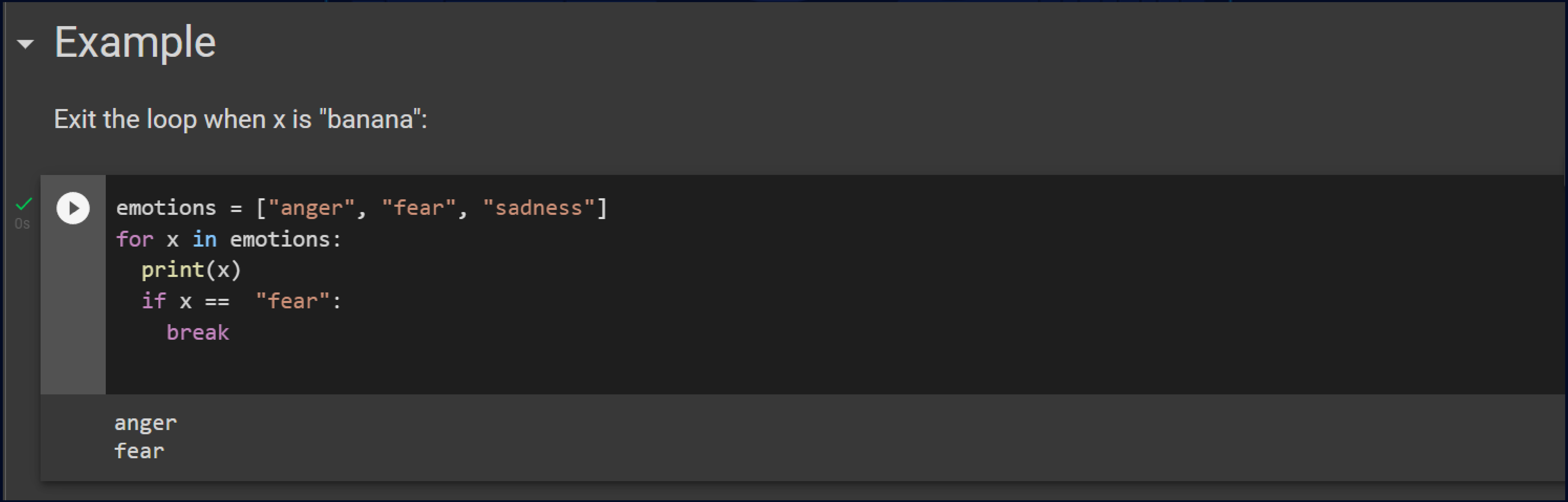
```
for x in "fear":
  print(x)
```

```
f
e
a
r
```

Python For Loops

The break Statement

We can break the loop before it loops through all of the items by using the break statement:



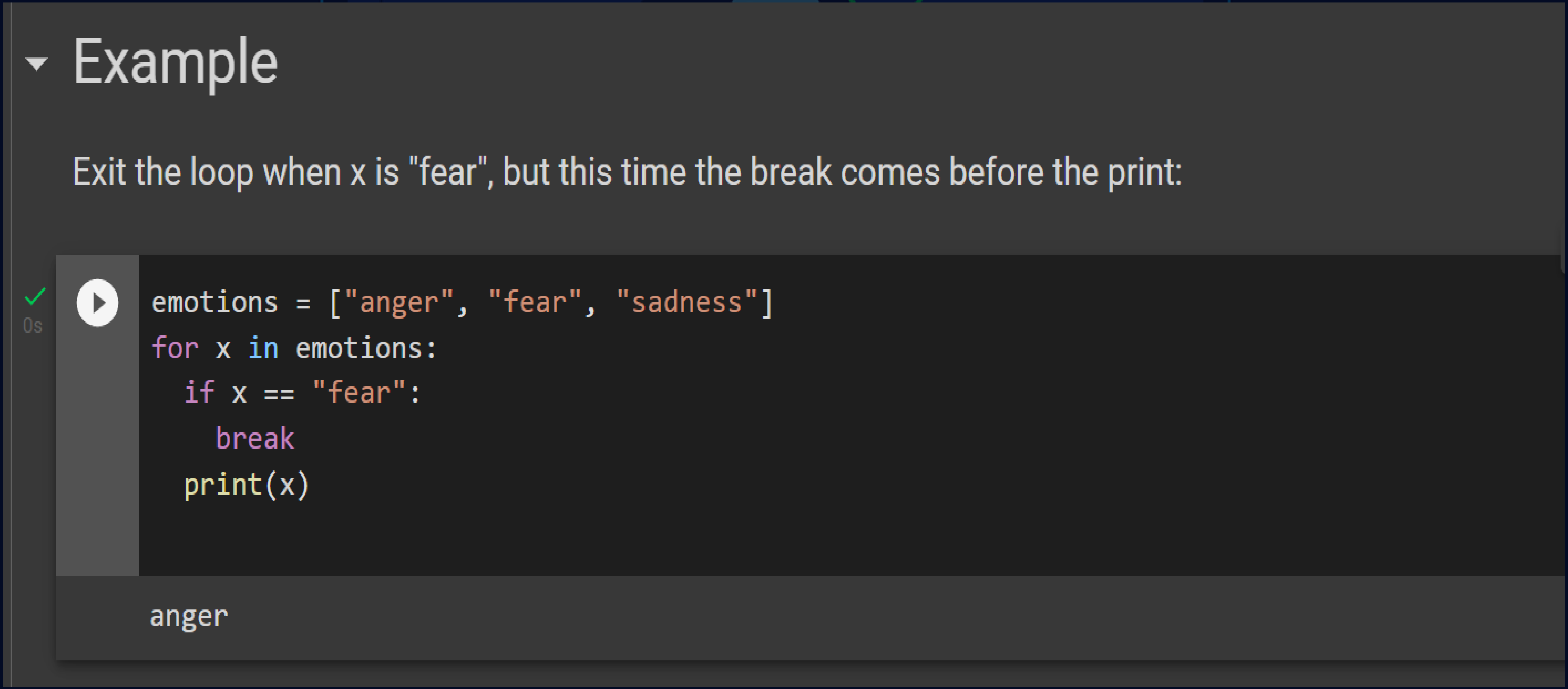
Example

Exit the loop when x is "banana":

```
emotions = ["anger", "fear", "sadness"]
for x in emotions:
  print(x)
  if x ==  "fear":
    break
```

```
anger
fear
```

Python For Loops



Example

Exit the loop when x is "fear", but this time the break comes before the print:

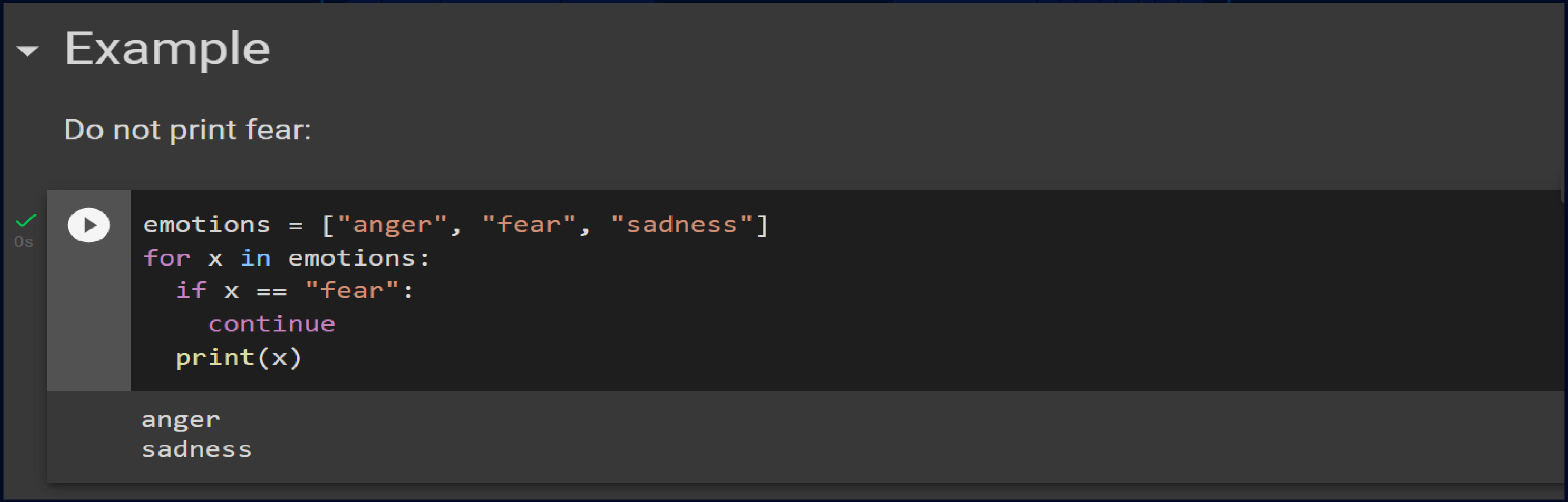
```
emotions = ["anger", "fear", "sadness"]
for x in emotions:
  if x == "fear":
    break
  print(x)
```

```
anger
```

Python For Loops

The continue Statement

We can use the continue statement to stop the current iteration of the loop and continue with the next:



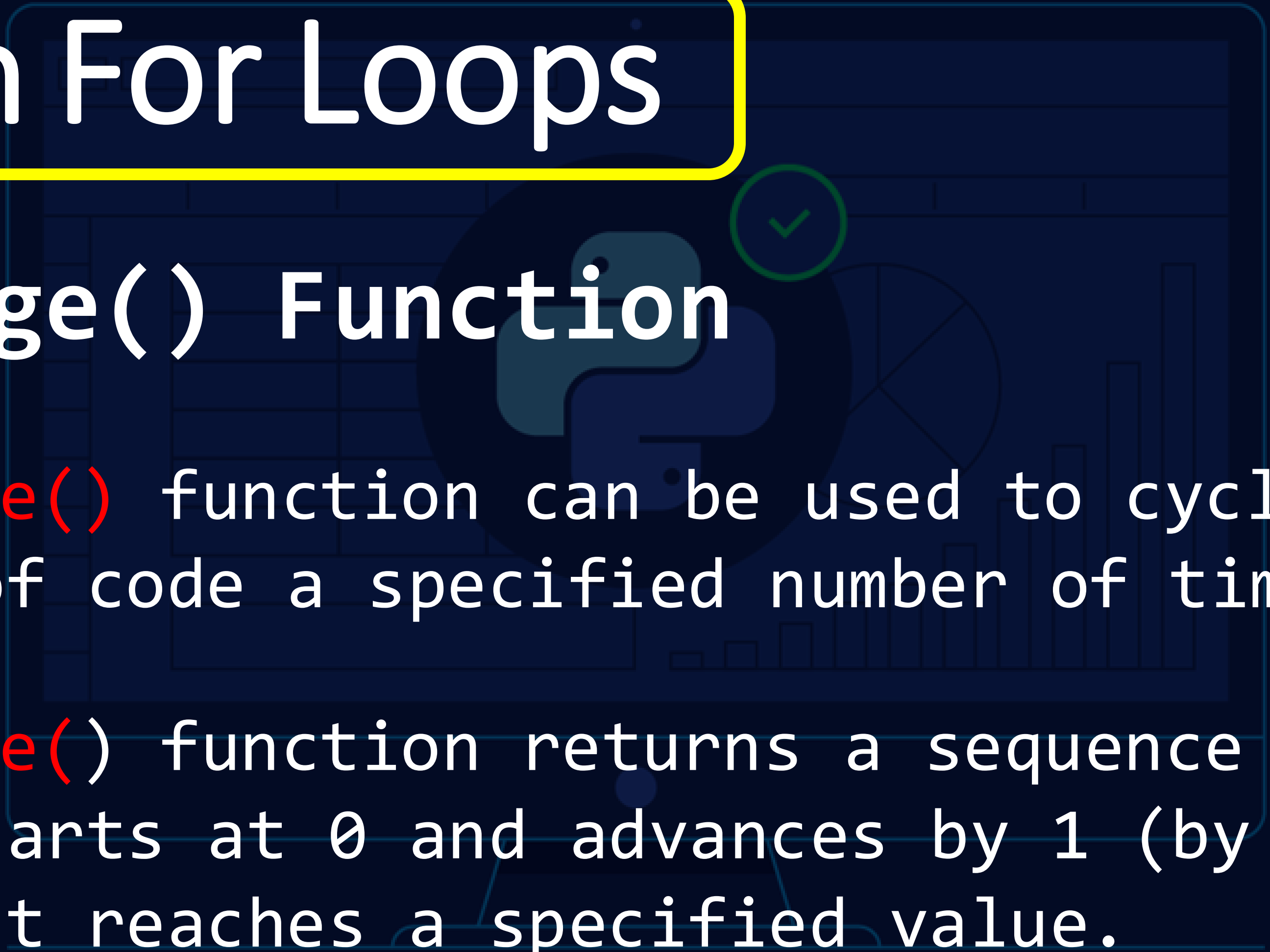
Example

Do not print fear:

```python
emotions = ["anger", "fear", "sadness"]
for x in emotions:
  if x == "fear":
    continue
  print(x)
```

```
anger
sadness
```

Python For Loops



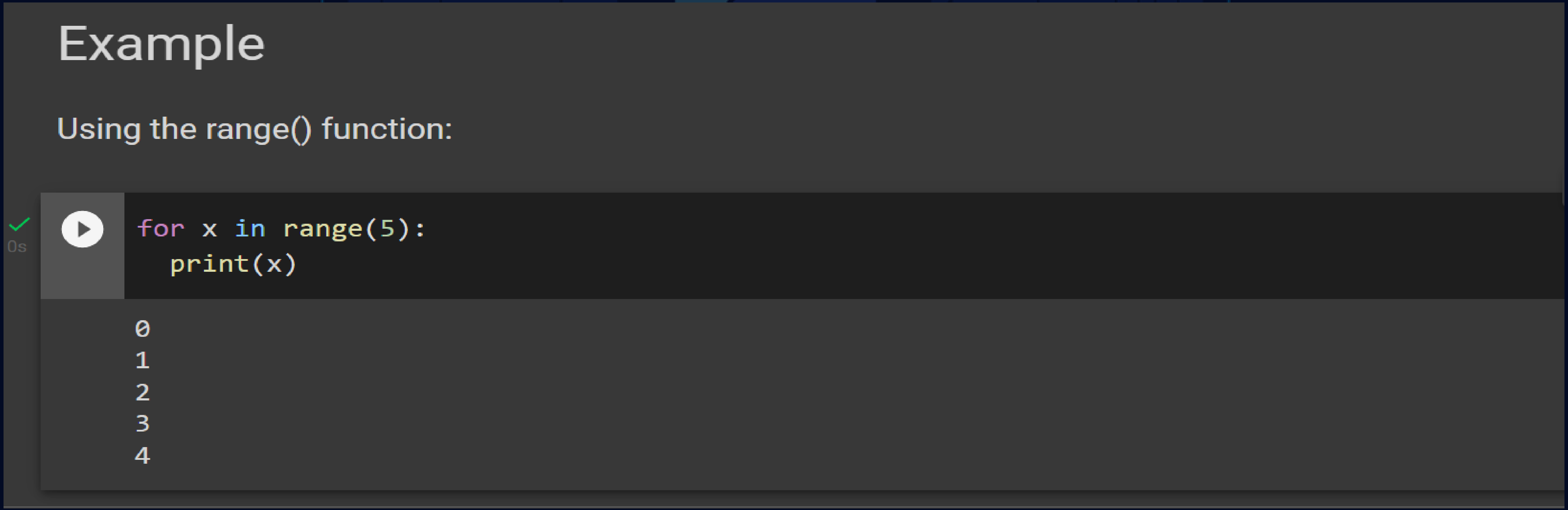
The range() Function

The range() function can be used to cycle through a set of code a specified number of times.

The range() function returns a sequence of numbers that starts at 0 and advances by 1 (by default) until it reaches a specified value.

Python For Loops

The range() Function



Example

Using the range() function:

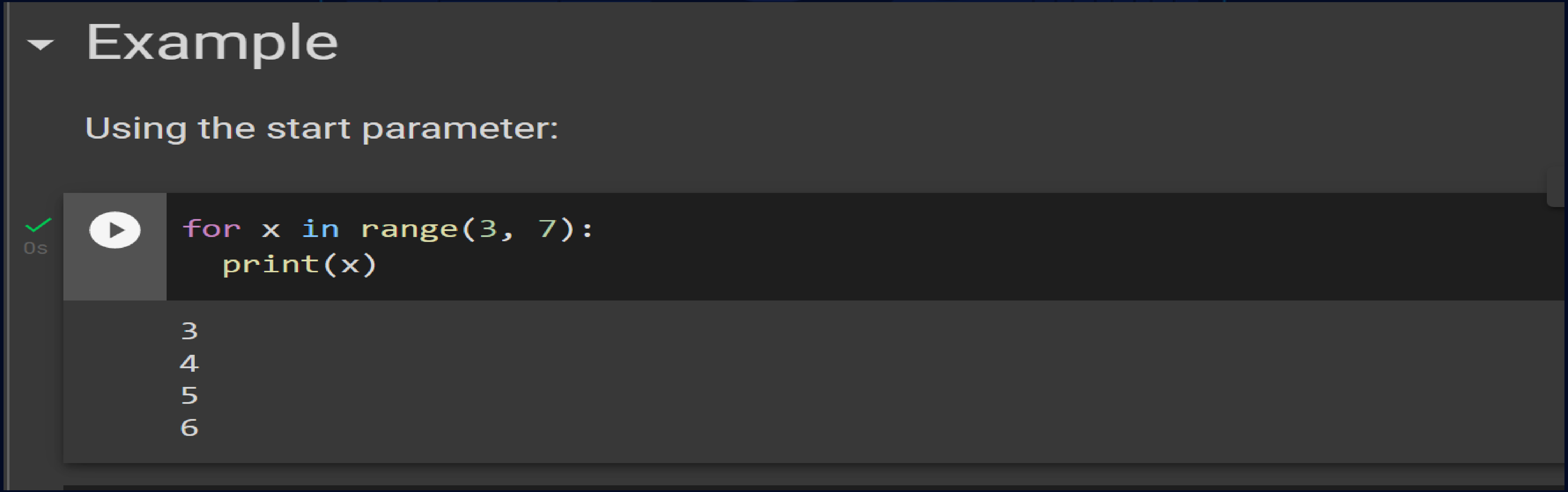
```
for x in range(5):
  print(x)
```

```
0
1
2
3
4
```

Note that range(5) is not the values of 0 to 5, but the values 0 to 4.

Python For Loops

The range() function defaults to 0 as a starting value, but you can change it by passing a parameter: range(3, 7), which means values from 3 to 7 (but not including 7):



Example

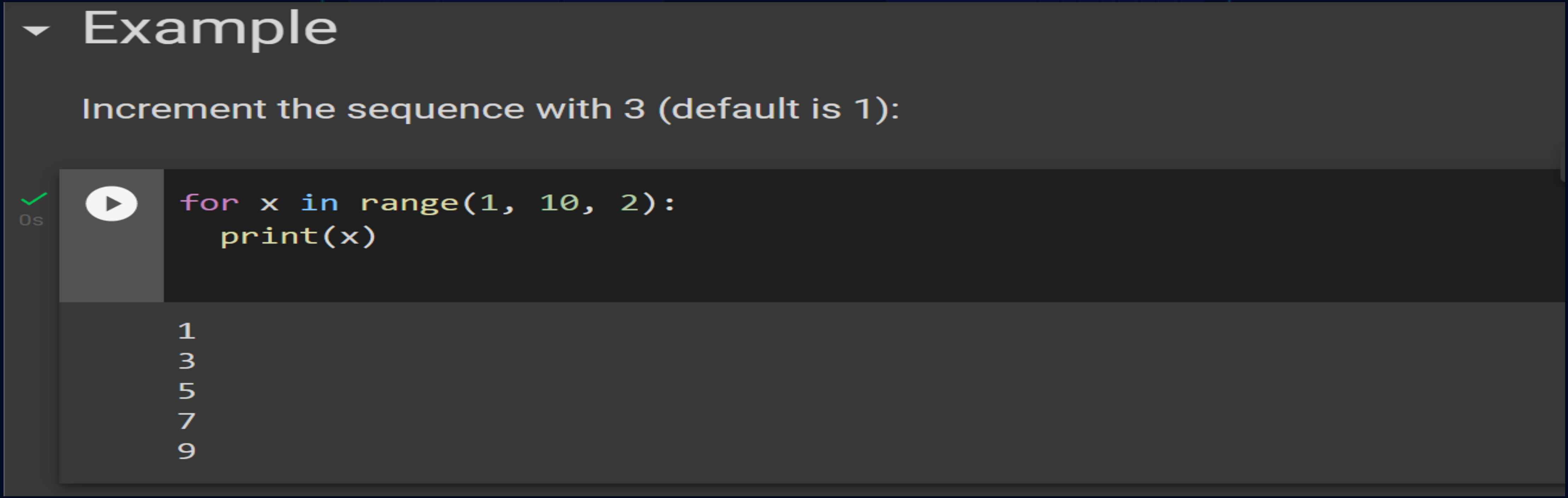
Using the start parameter:

```
for x in range(3, 7):
  print(x)
```

```
3
4
5
6
```

Python For Loops

The range() function by default increments the sequence by one, but a third parameter can be used to specify the increment value: range(1, 10, 2):



Example

Increment the sequence with 3 (default is 1):

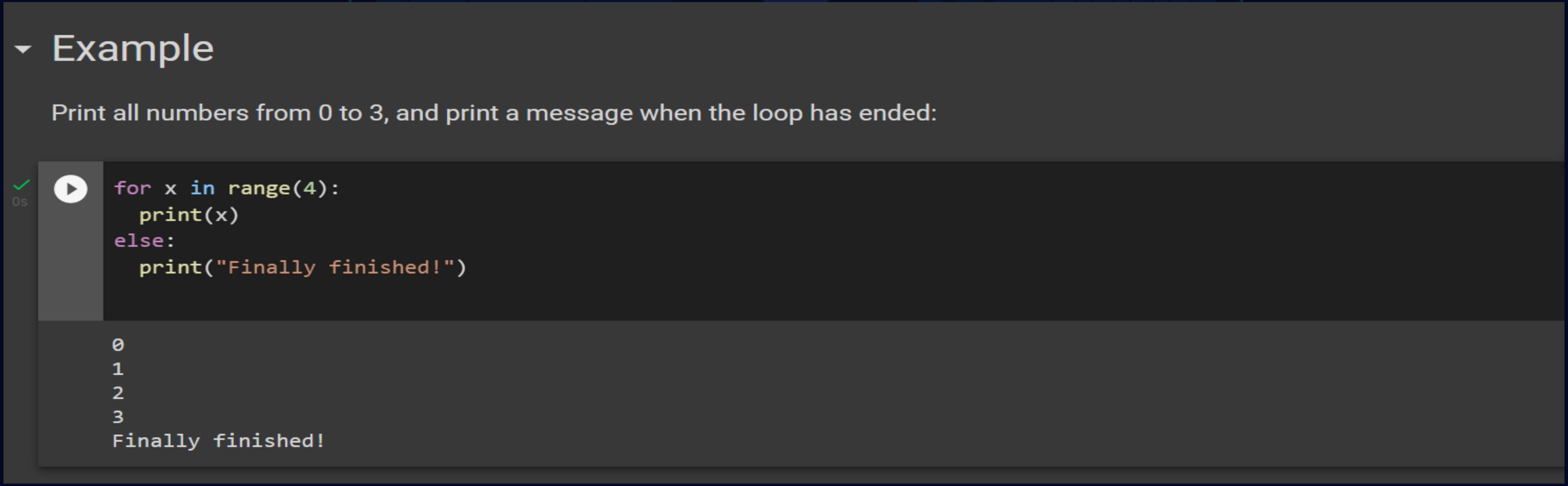
```python
for x in range(1, 10, 2):
  print(x)
```

```
1
3
5
7
9
```

Python For Loops

Else in For Loop

The else keyword in a for loop specifies a block of code to be executed when the loop is finished:



Example

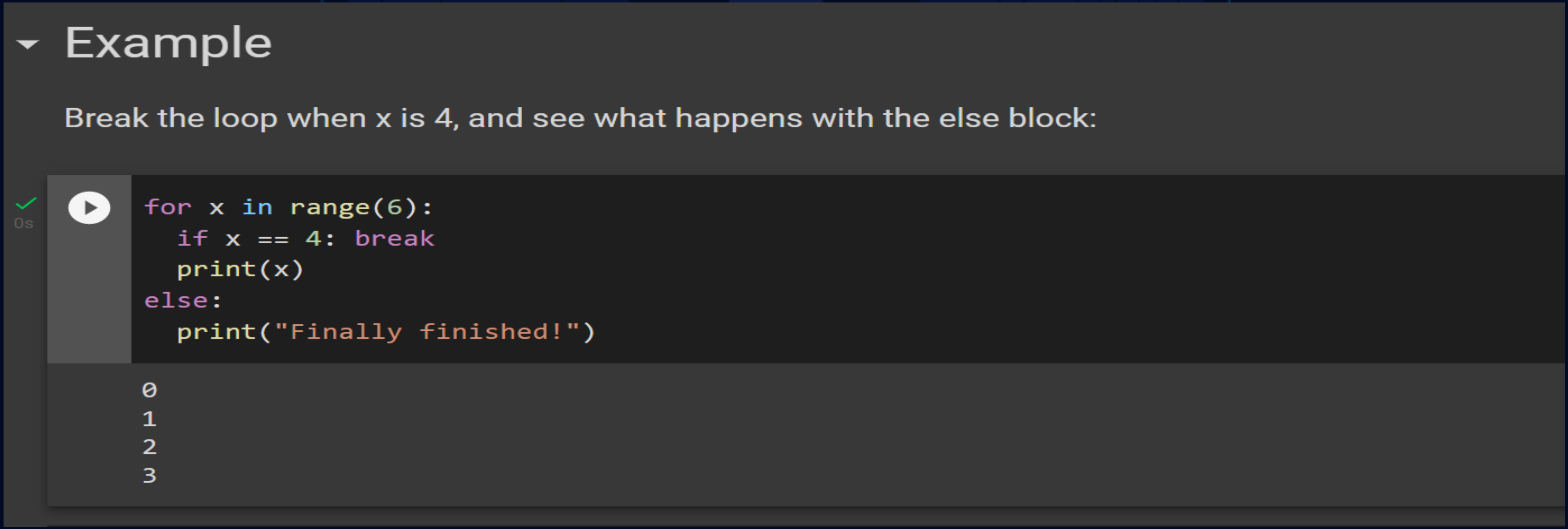
Print all numbers from 0 to 3, and print a message when the loop has ended:

```python
for x in range(4):
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
Finally finished!
```

Python For Loops

Note: If the loop is interrupted by a break statement, the else block will not be executed.



Example

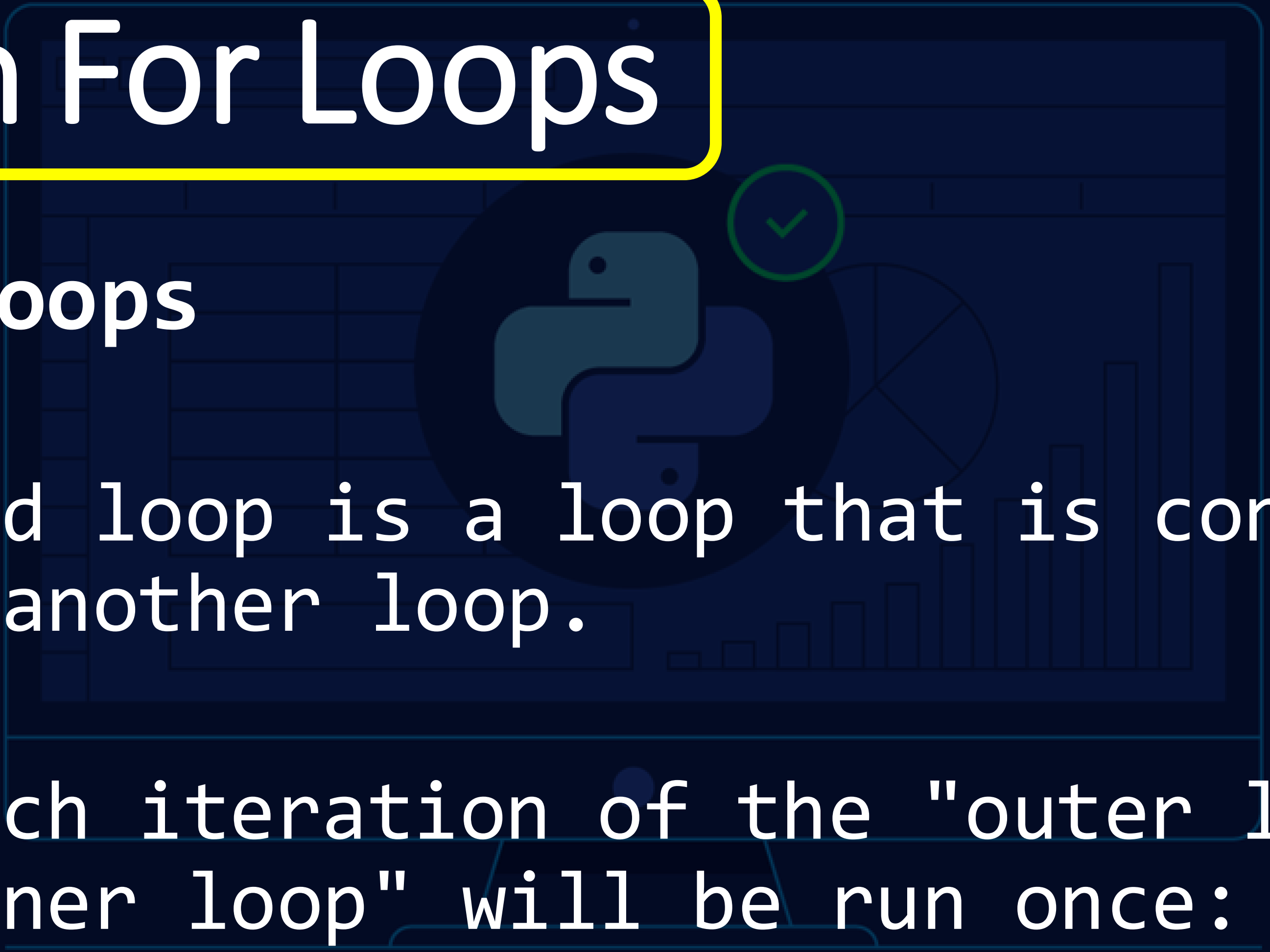
Break the loop when x is 4, and see what happens with the else block:

```python
for x in range(6):
  if x == 4: break
  print(x)
else:
  print("Finally finished!")
```

```
0
1
2
3
```

Python For Loops

Nested Loops

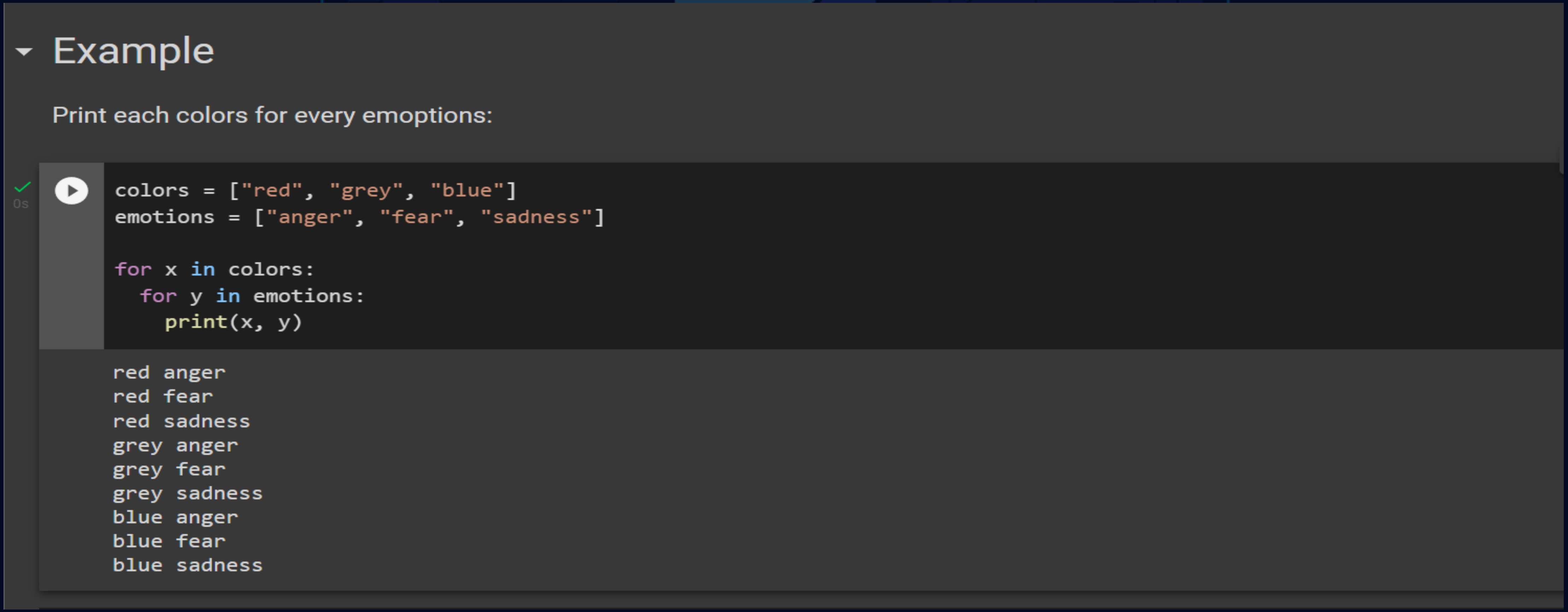


A nested loop is a loop that is contained within another loop.

For each iteration of the "outer loop," the "inner loop" will be run once:

Python For Loops

Nested Loops



Example

Print each colors for every emoptions:

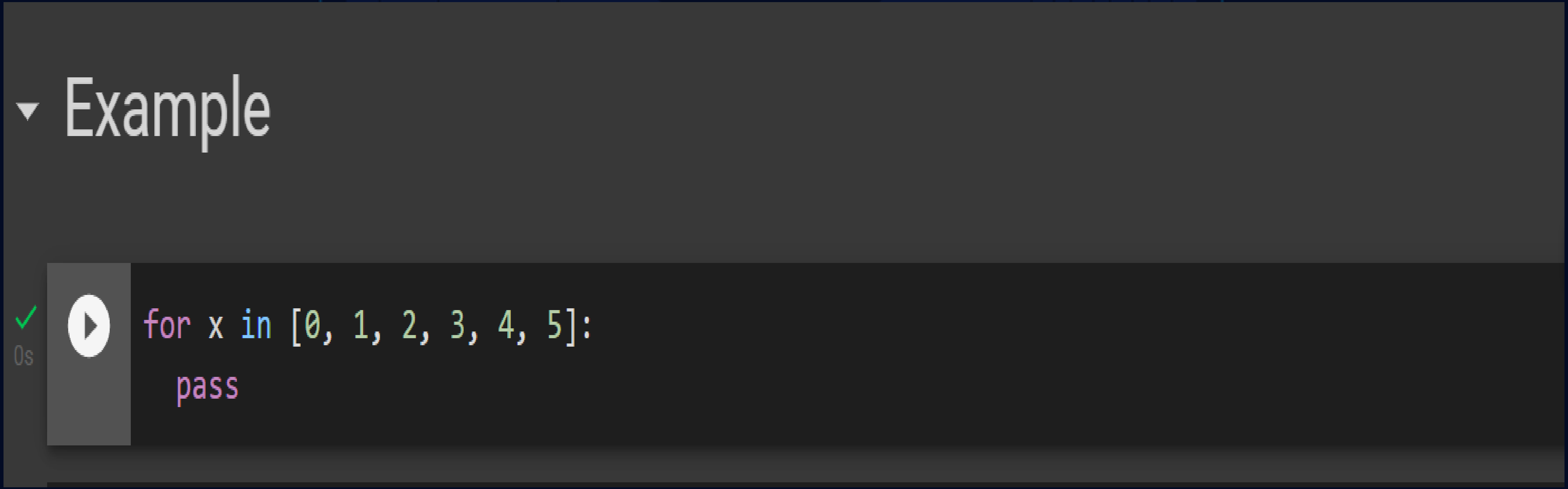
```
colors = ["red", "grey", "blue"]
emotions = ["anger", "fear", "sadness"]

for x in colors:
  for y in emotions:
    print(x, y)
```

```
red anger
red fear
red sadness
grey anger
grey fear
grey sadness
blue anger
blue fear
blue sadness
```

Python For Loops

The pass Statement

For loops cannot be empty, but if you have a for loop with no content, use the pass statement to prevent an error.

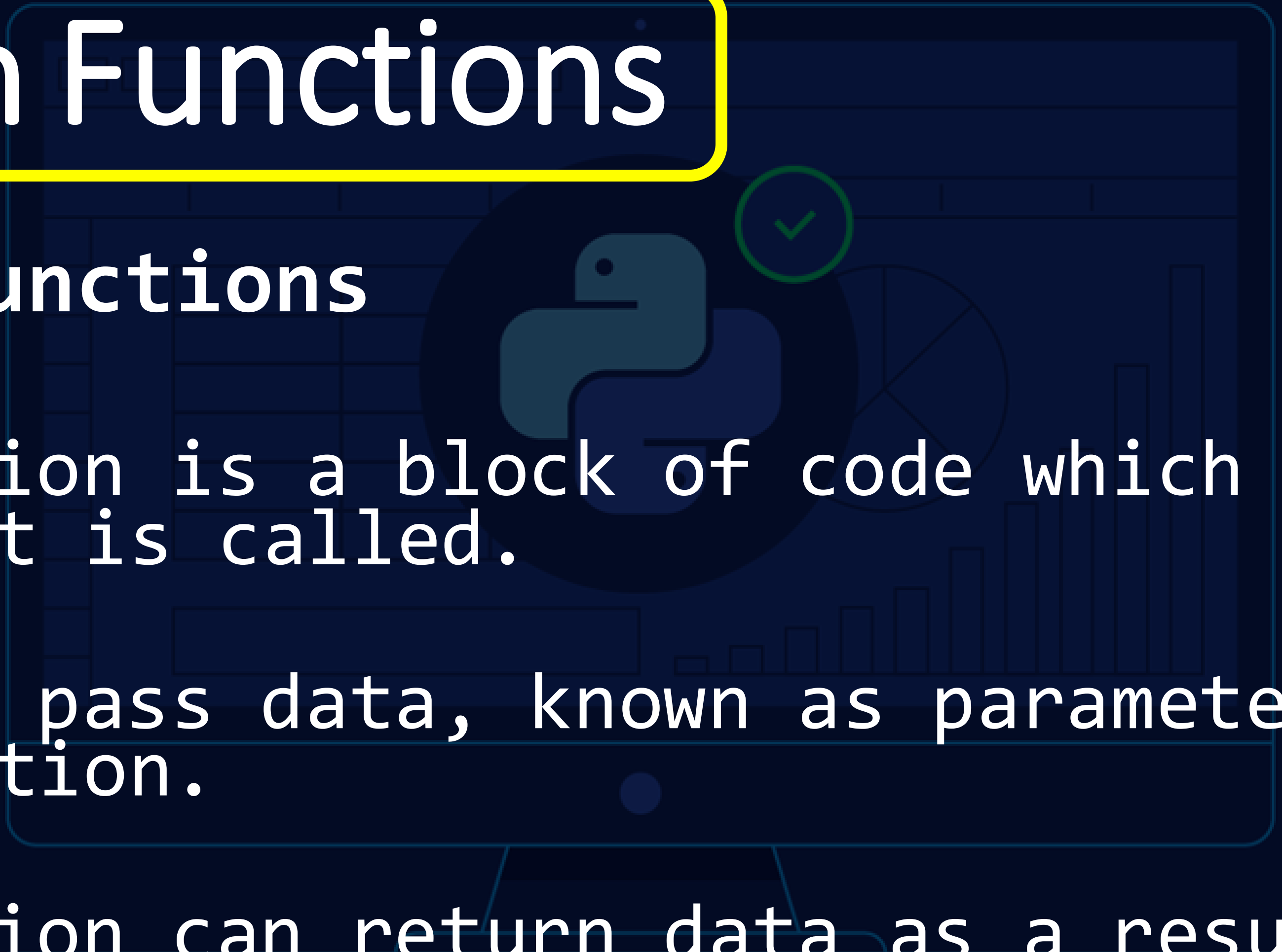


Example

```
for x in [0, 1, 2, 3, 4, 5]:
  pass
```

Python Functions

Python Functions



A function is a block of code which only runs when it is called.

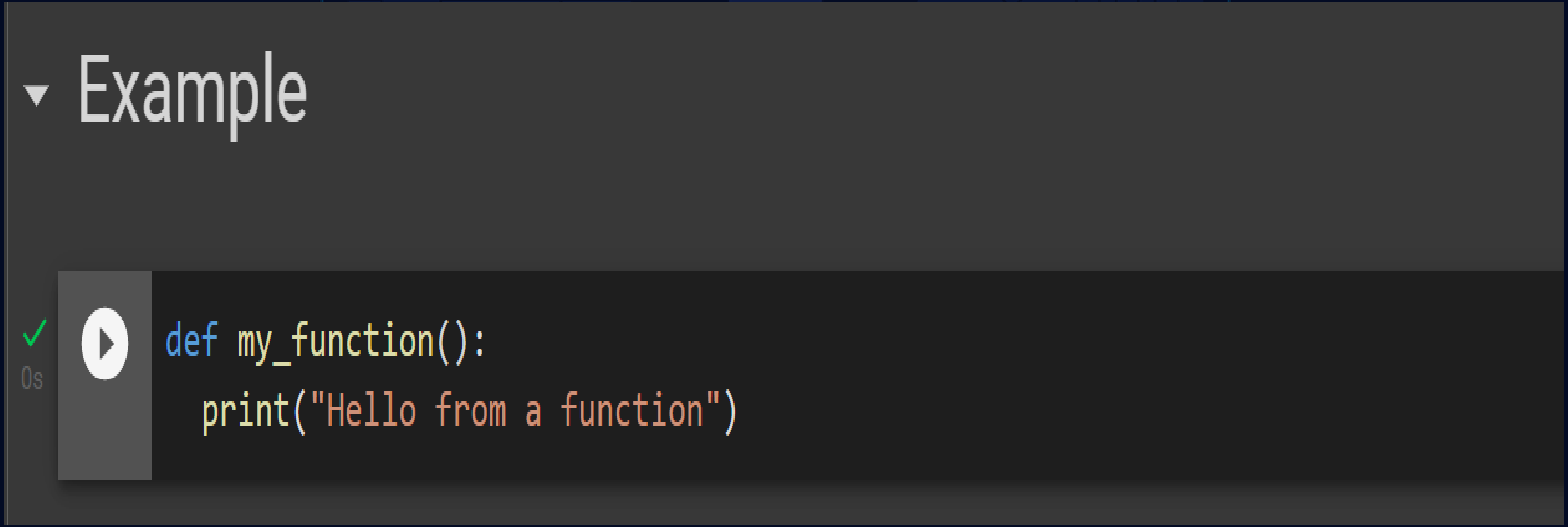
You can pass data, known as parameters, into a function.

A function can return data as a result.

Python Functions

Creating a Function

The def keyword is used to define a function in Python:



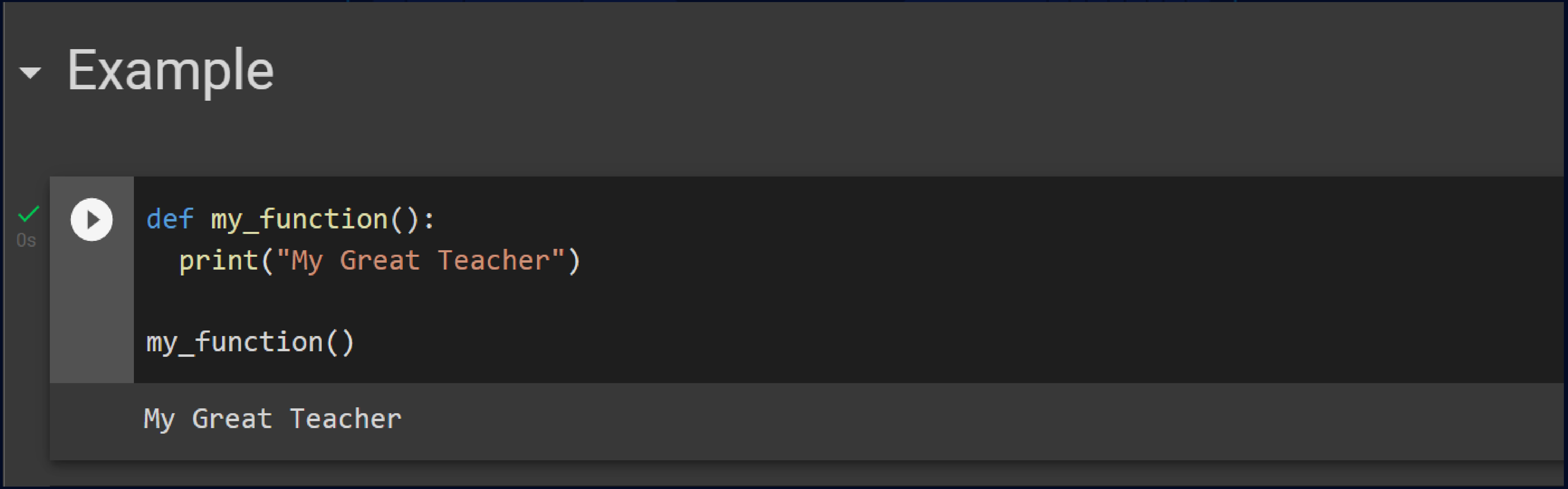
Example

```python
def my_function():
  print("Hello from a function")
```

Python Functions

Calling a Function

To invoke a function, use its name followed by parenthesis:

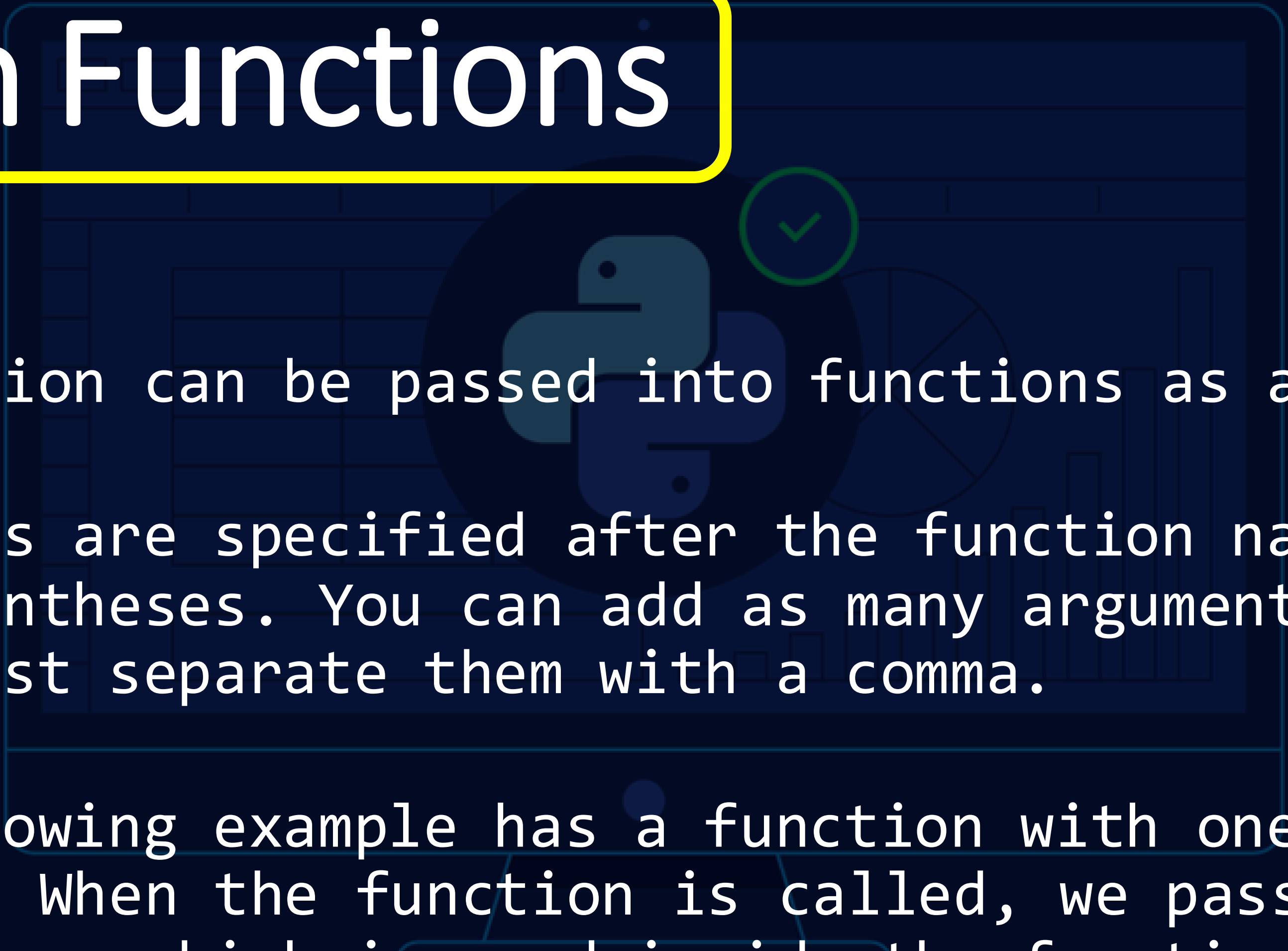


Example

```python
def my_function():
  print("My Great Teacher")

my_function()
```

```
My Great Teacher
```

Python Functions



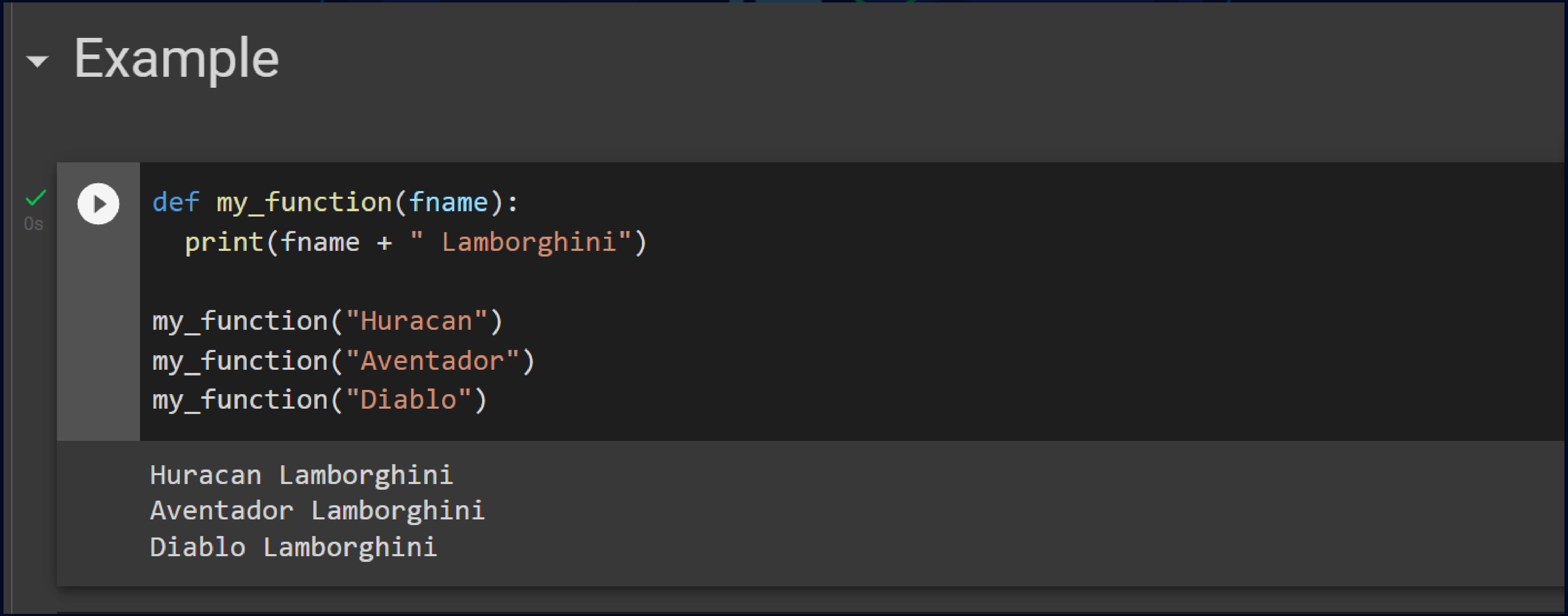
Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Python Functions



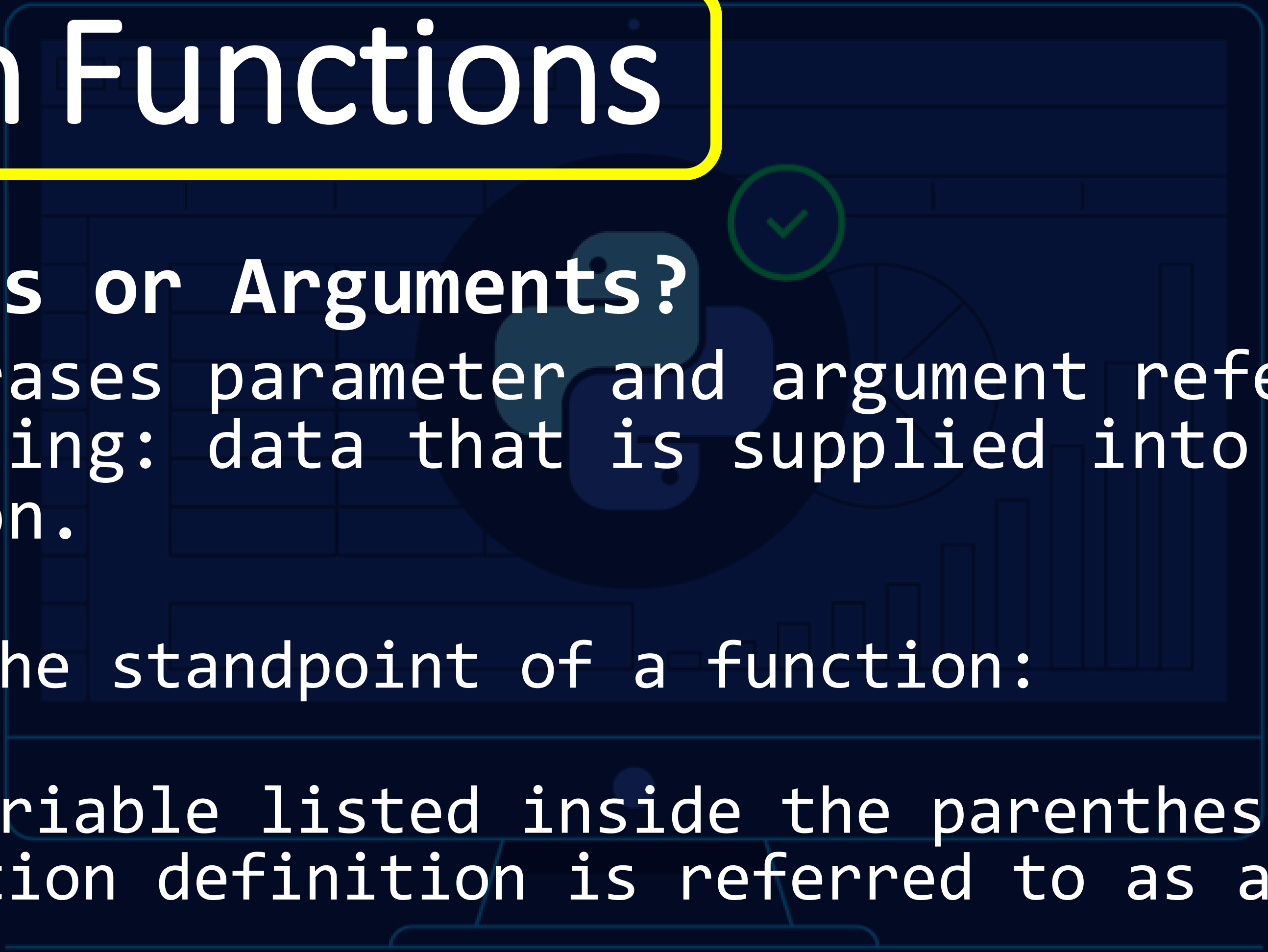
Example

```
def my_function(fname):
  print(fname + " Lamborghini")

my_function("Huracan")
my_function("Aventador")
my_function("Diablo")
```

```
Huracan Lamborghini
Aventador Lamborghini
Diablo Lamborghini
```

In Python documentation, arguments are frequently abbreviated as args.

Python Functions



Parameters or Arguments?

The phrases parameter and argument refer to the same thing: data that is supplied into a function.

From the standpoint of a function:

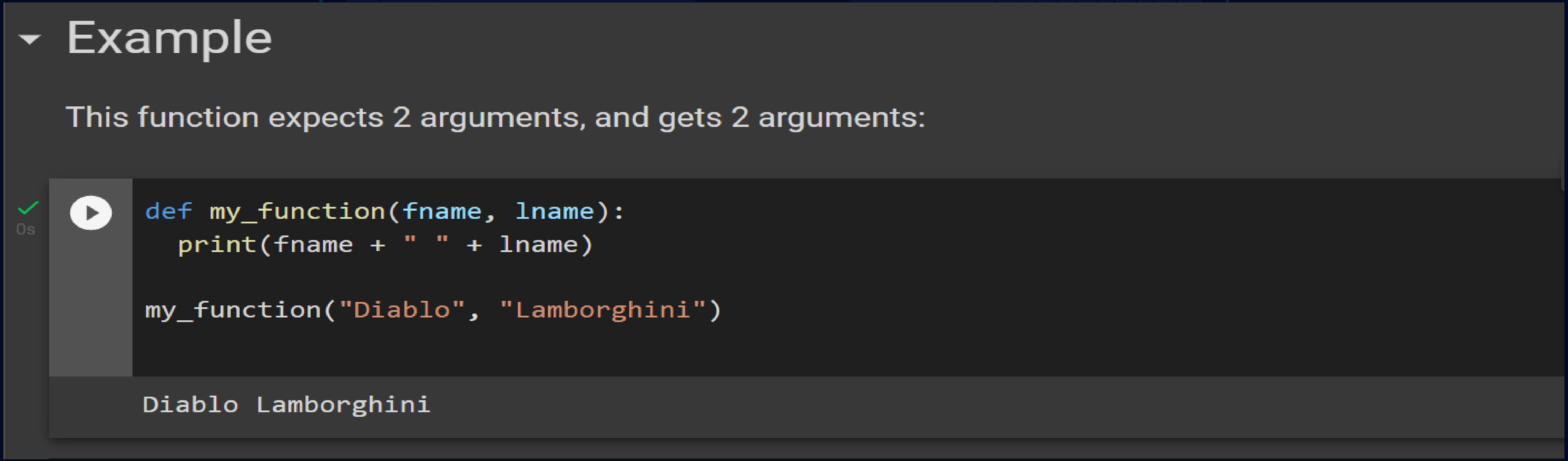
The variable listed inside the parentheses in the function definition is referred to as a parameter.

An argument is the value passed to the function when it is invoked.

Python Functions

Number of Arguments

A function must be called with the right number of parameters by default. That is, if your function expects two parameters, you must call it with two arguments, not more or fewer.



Example

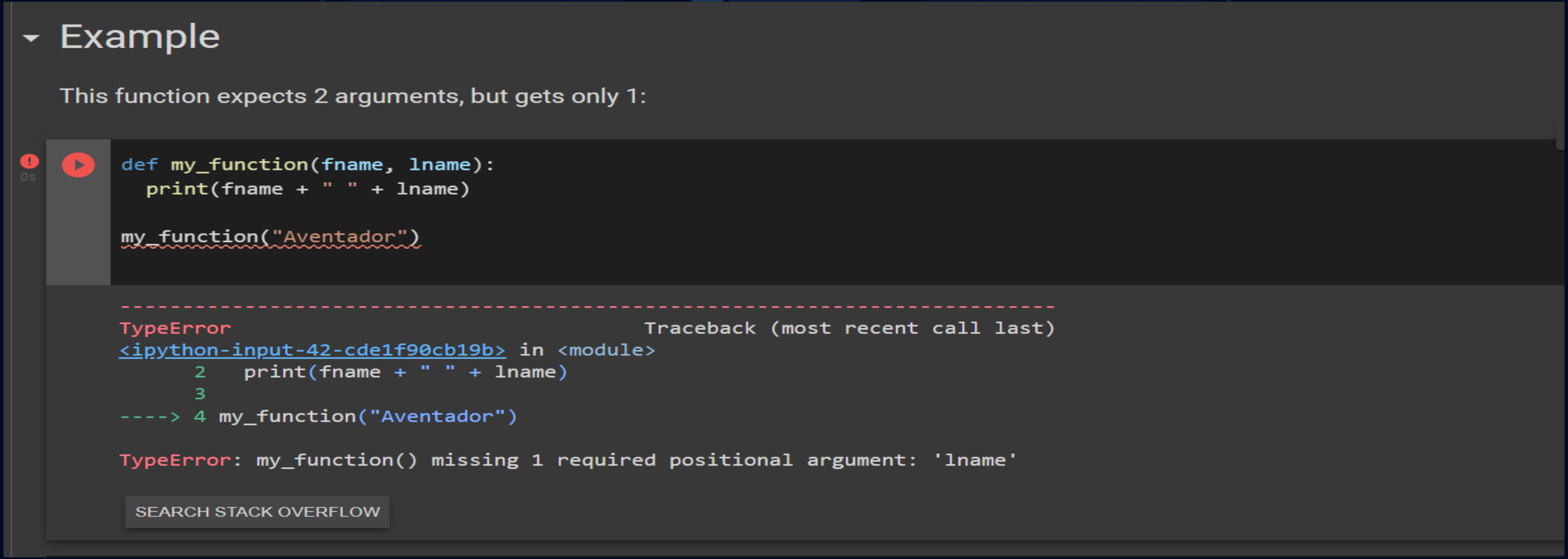
This function expects 2 arguments, and gets 2 arguments:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Diablo", "Lamborghini")
```

```
Diablo Lamborghini
```

Python Functions

If you try to call the function with one or three arguments, you will get the following error:



Example

This function expects 2 arguments, but gets only 1:

```
def my_function(fname, lname):
  print(fname + " " + lname)

my_function("Aventador")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-42-cde1f90cb19b> in <module>
      2   print(fname + " " + lname)
      3 
----> 4 my_function("Aventador")

TypeError: my_function() missing 1 required positional argument: 'lname'
```

Python Functions

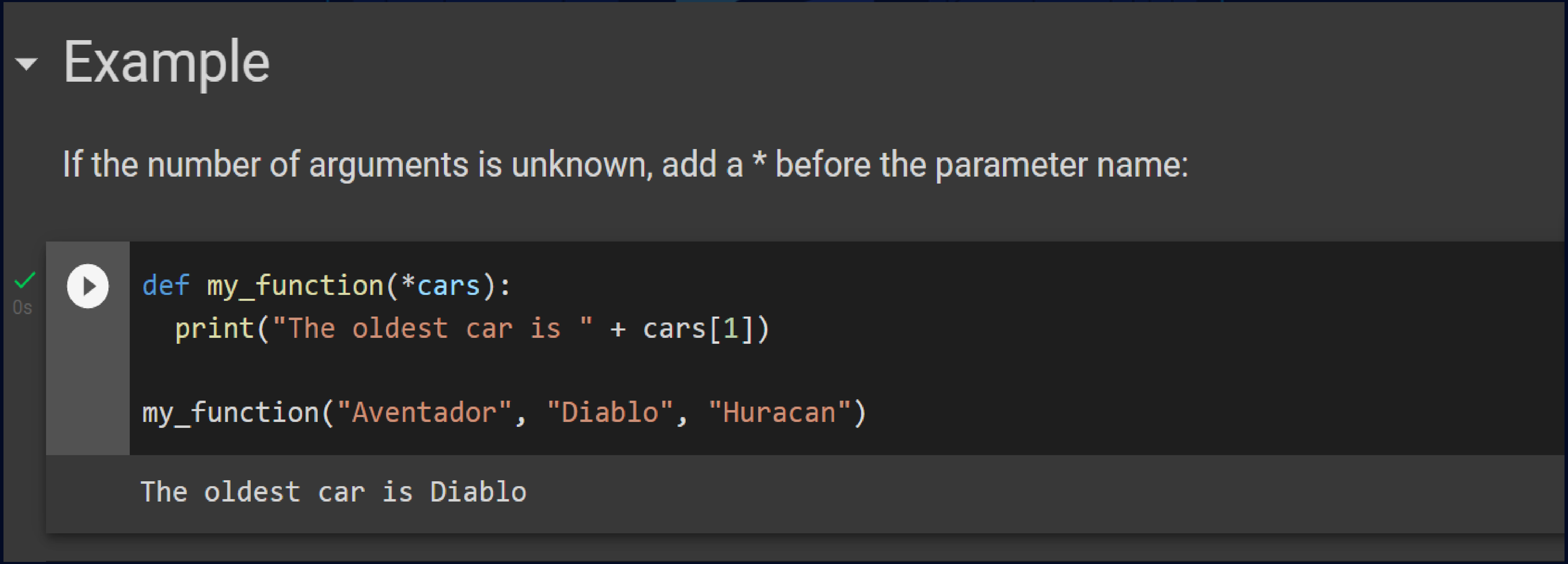
Arbitrary Arguments, *args

If you don't know how many arguments your function will get, add a * before the parameter name in the function specification.

Like a result, the function will receive a tuple of parameters and will be able to retrieve the elements as follows:

Python Functions

Arbitrary Arguments, *args



Example

If the number of arguments is unknown, add a * before the parameter name:

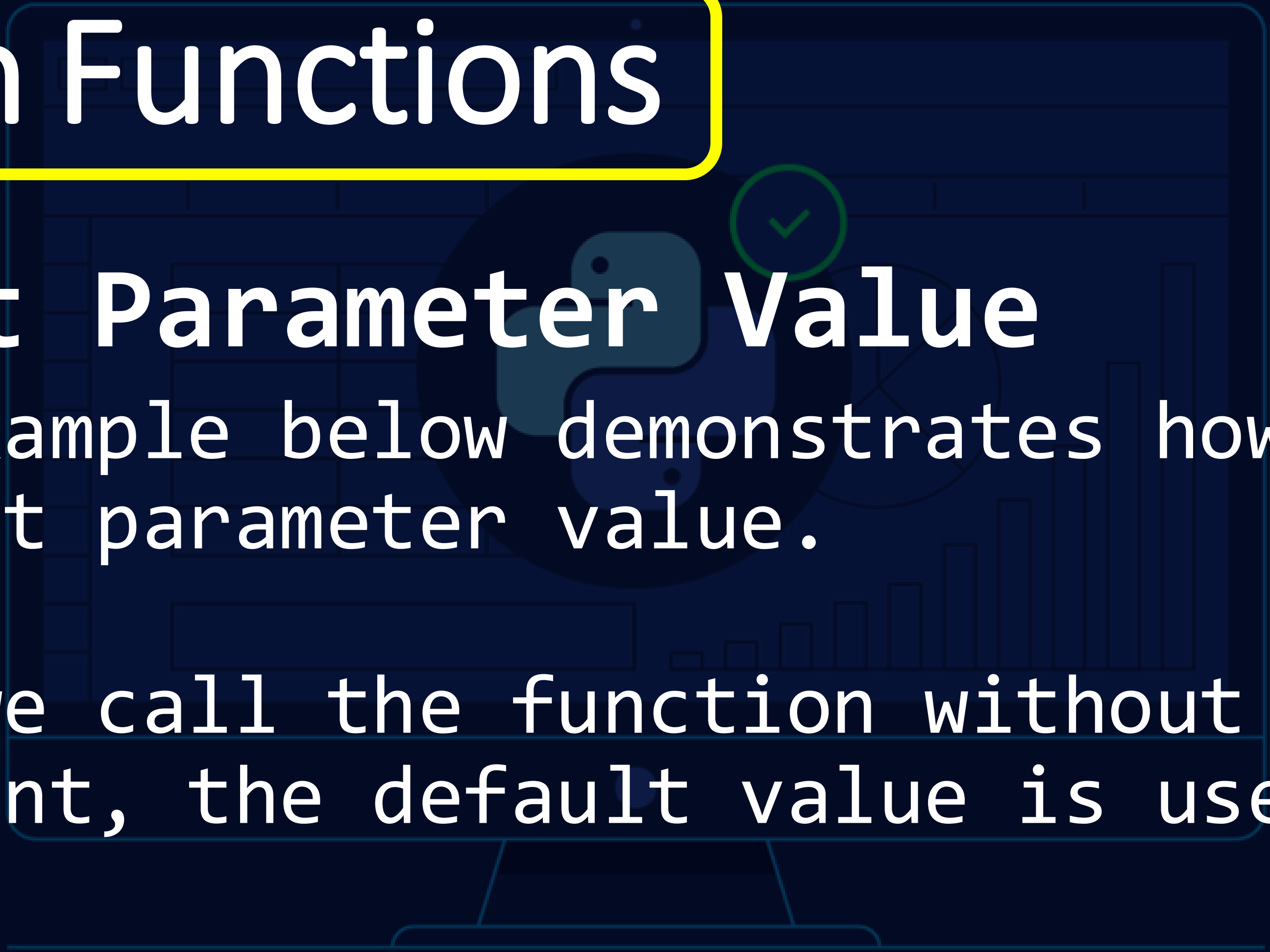
```
def my_function(*cars):
  print("The oldest car is " + cars[1])

my_function("Aventador", "Diablo", "Huracan")
```

```
The oldest car is Diablo
```

Arbitrary arguments are frequently abbreviated to *args in Python documentation.

Python Functions

Default Parameter Value

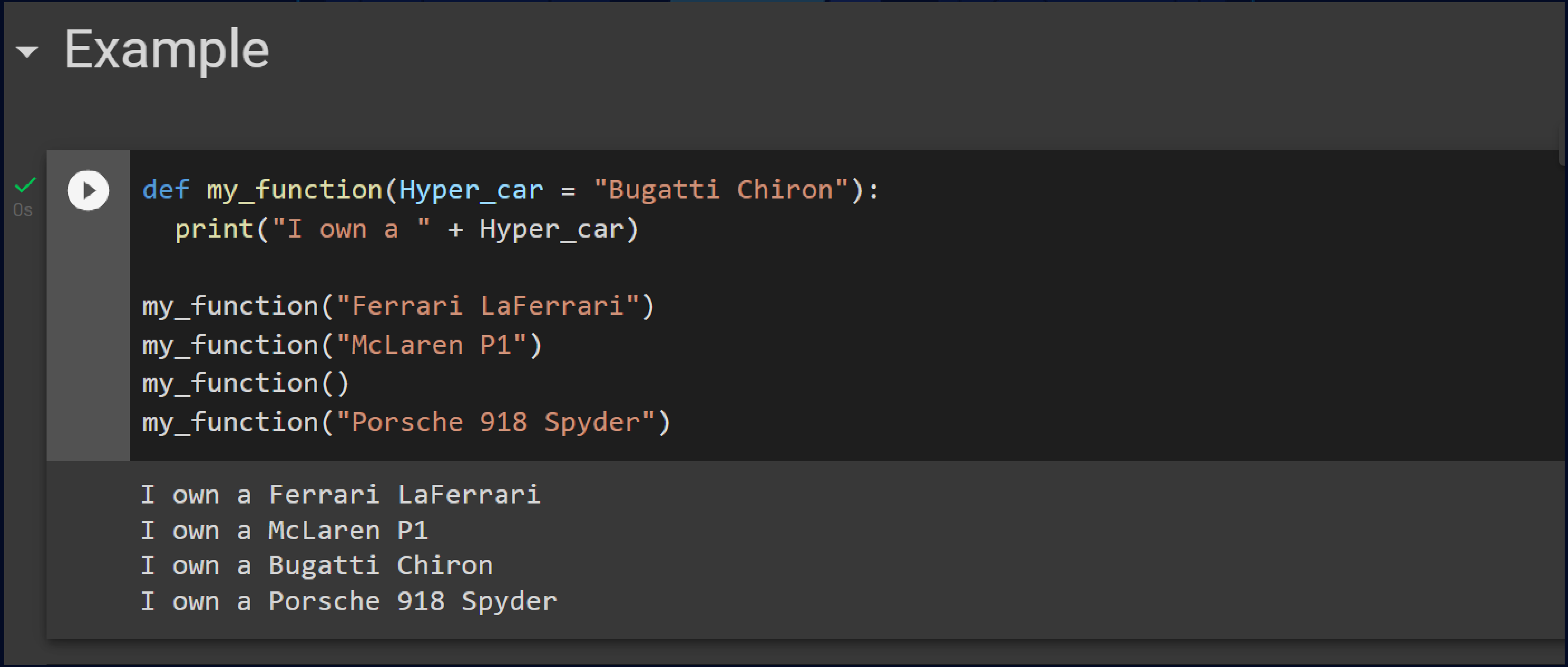


The example below demonstrates how to use a default parameter value.

When we call the function without an argument, the default value is used:

Python Functions

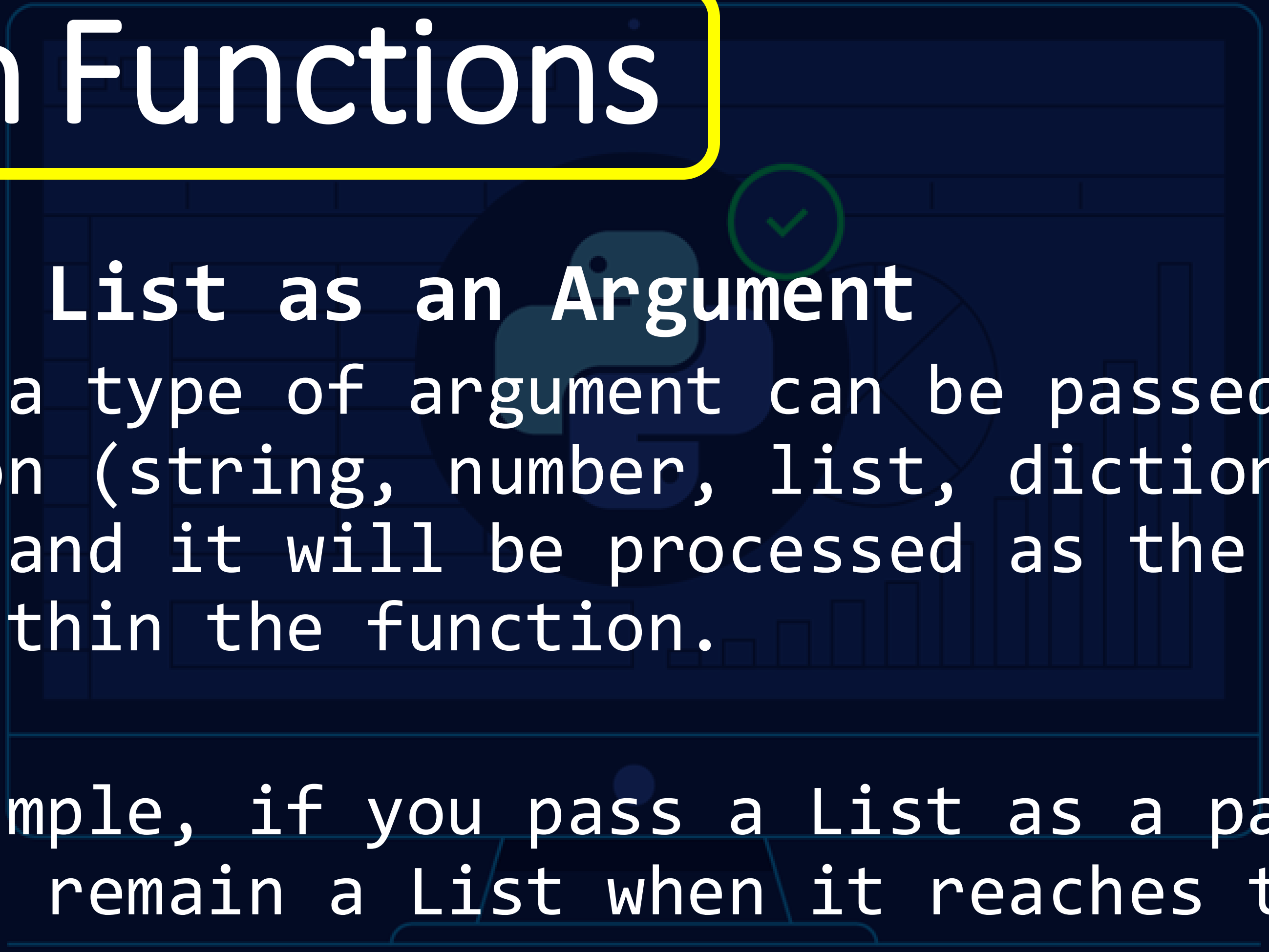
Default Parameter Value



```python
def my_function(Hyper_car = "Bugatti Chiron"):
  print("I own a " + Hyper_car)

my_function("Ferrari LaFerrari")
my_function("McLaren P1")
my_function()
my_function("Porsche 918 Spyder")
```

```
I own a Ferrari LaFerrari
I own a McLaren P1
I own a Bugatti Chiron
I own a Porsche 918 Spyder
```

Python Functions



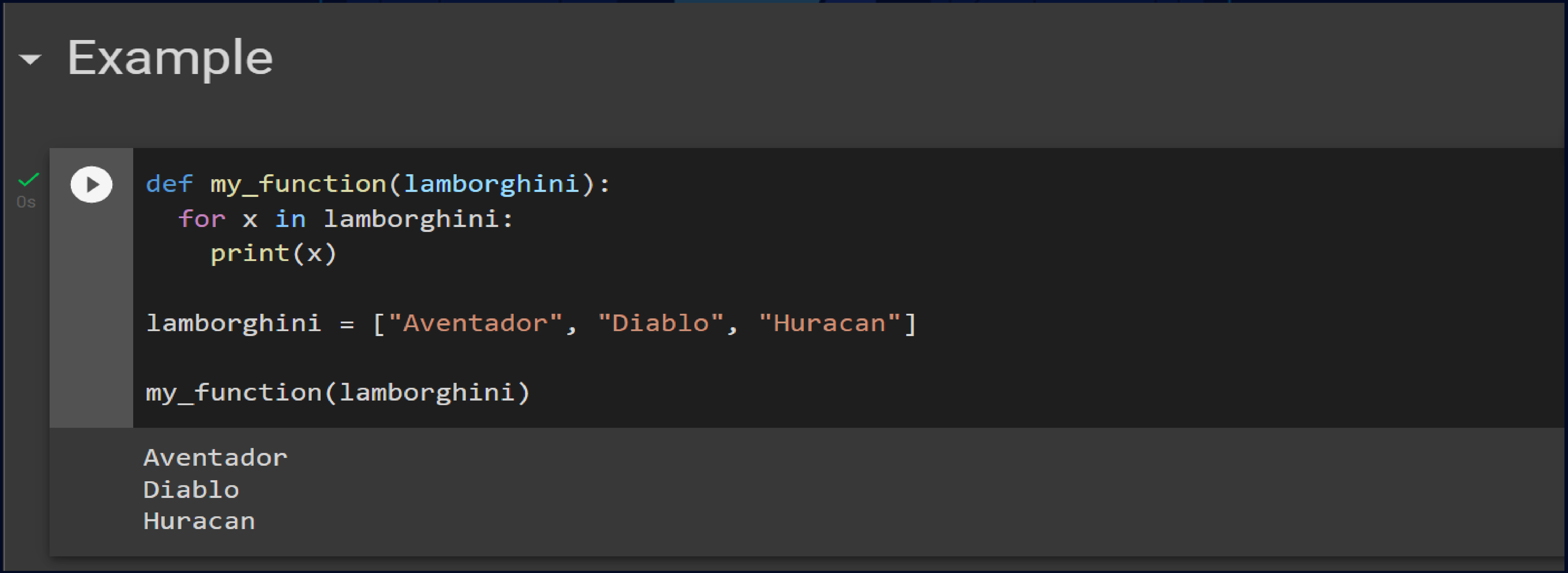
Passing a List as an Argument

Any data type of argument can be passed to a function (string, number, list, dictionary, etc.), and it will be processed as the same data type within the function.

For example, if you pass a List as a parameter, it will remain a List when it reaches the function:

Python Functions

Passing a List as an Argument



Example

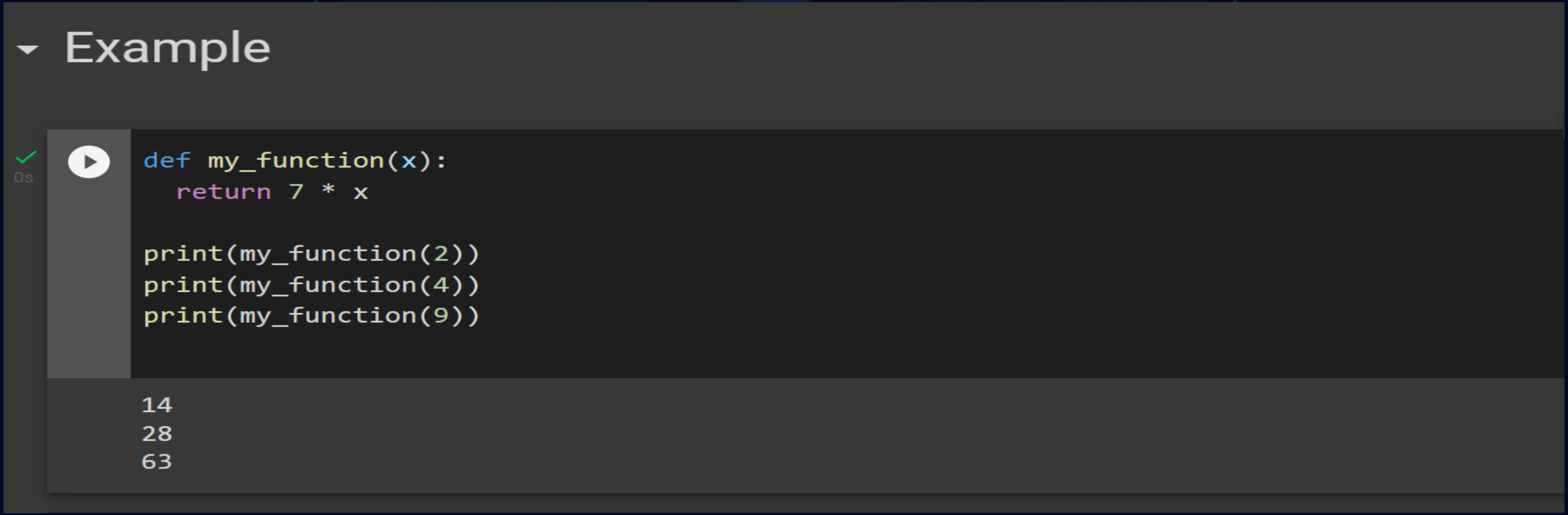
```python
def my_function(lamborghini):
  for x in lamborghini:
    print(x)

lamborghini = ["Aventador", "Diablo", "Huracan"]

my_function(lamborghini)
```

```
Aventador
Diablo
Huracan
```

Python Functions

Return Values

Use the return statement to allow a function to return a value:



Example

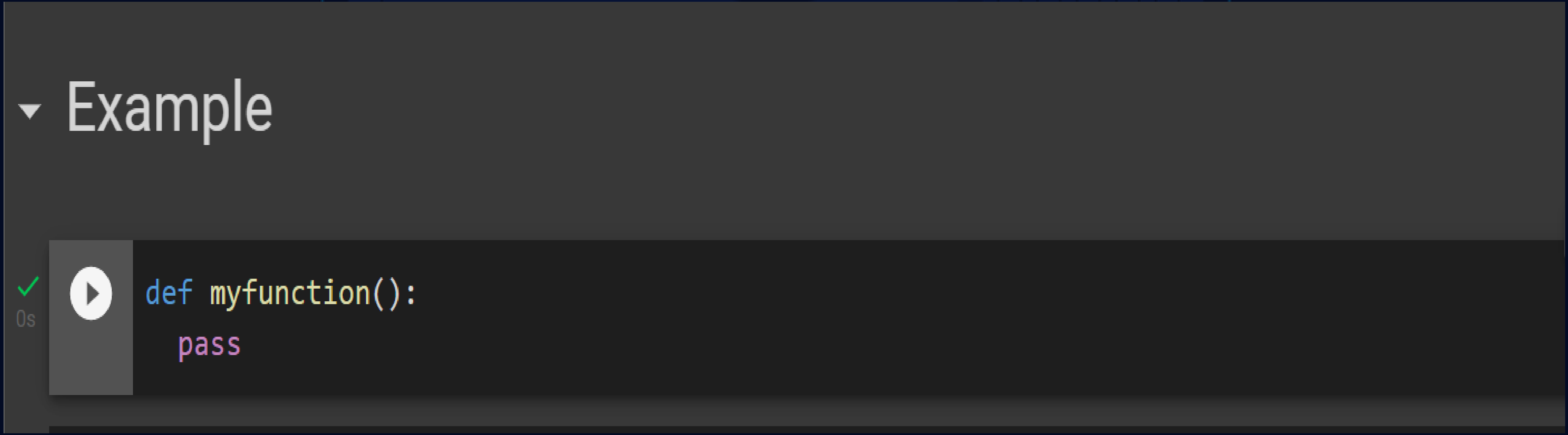
```
def my_function(x):
  return 7 * x

print(my_function(2))
print(my_function(4))
print(my_function(9))
```

```
14
28
63
```

Python Functions

The pass Statement

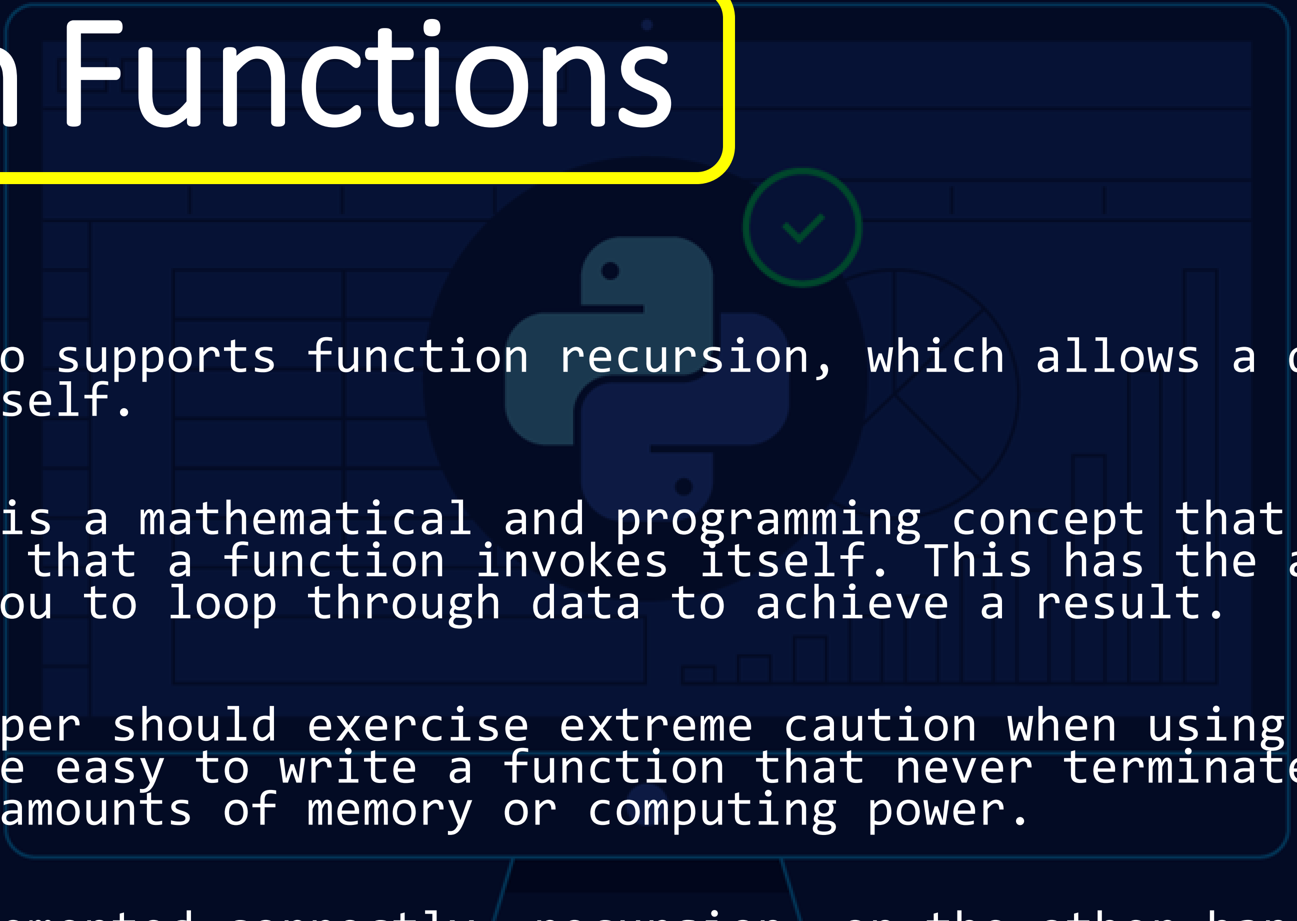
Function definitions cannot be empty, but if you have a function definition with no content for some reason, include the pass statement to avoid an error.



Example

```
def myfunction():
  pass
```

Python Functions



Recursion

- Python also supports function recursion, which allows a defined function to call itself.

- Recursion is a mathematical and programming concept that is widely used. It denotes that a function invokes itself. This has the advantage of allowing you to loop through data to achieve a result.

- The developer should exercise extreme caution when using recursion since it is quite easy to write a function that never terminates or consumes excessive amounts of memory or computing power.

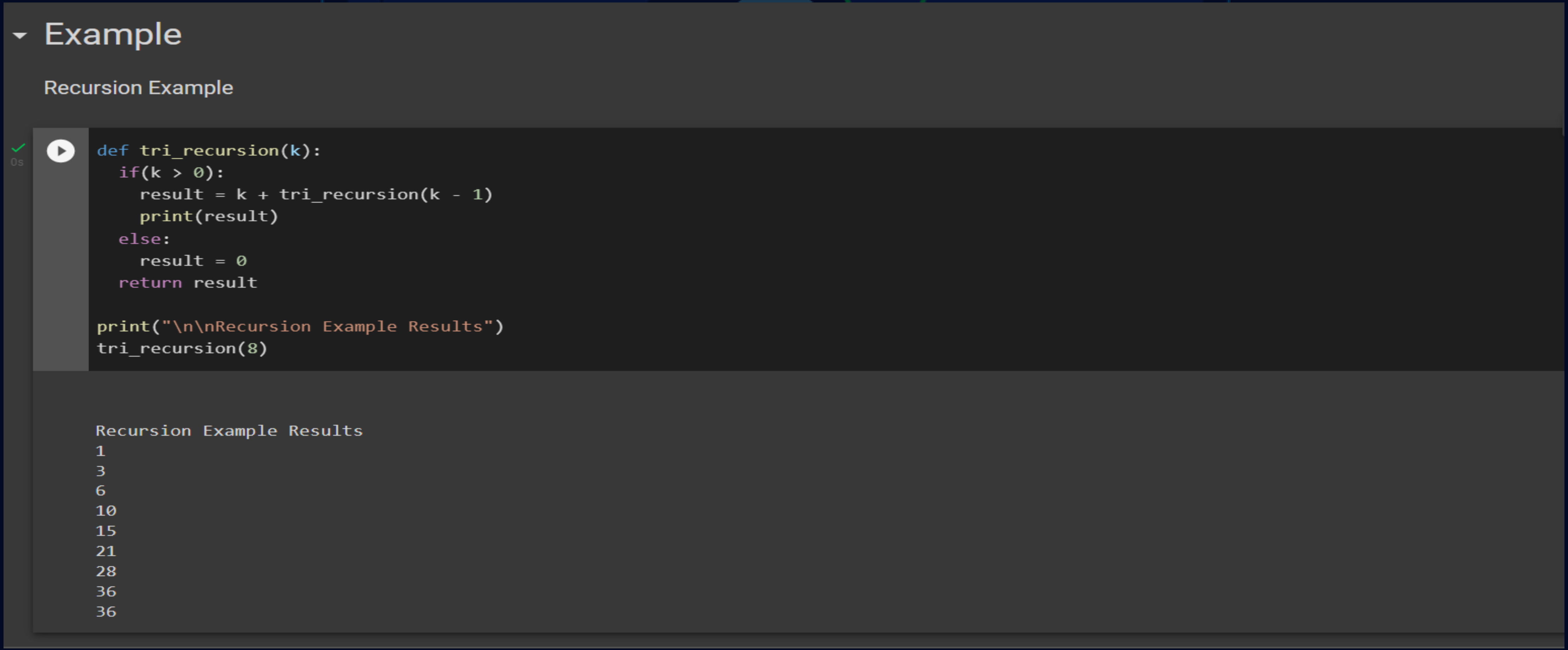
- When implemented correctly, recursion, on the other hand, can be a tremendously efficient and mathematically elegant technique to programming.

Python Functions

In this case, tri recursion() is a function defined to call itself ("recurse"). As data, we use the k variable, which decrements (-1) each time we recurse. When the condition is not greater than 0, the recursion ends.

It may take some time for a new developer to figure out how this works; the best way to find out is to test and modify it.

Python Functions



Example

Recursion Example

```
def tri_recursion(k):
  if(k > 0):
    result = k + tri_recursion(k - 1)
    print(result)
  else:
    result = 0
  return result

print("\n\nRecursion Example Results")
tri_recursion(8)
```

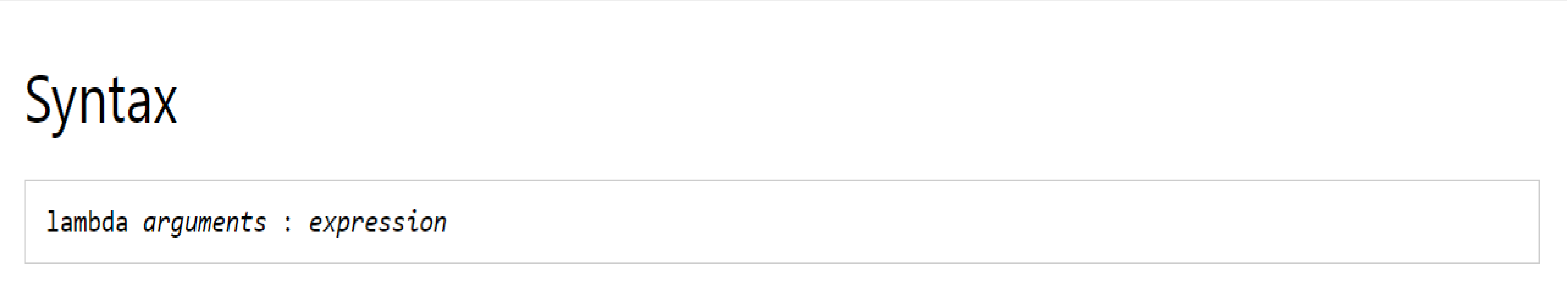
```


Recursion Example Results
1
3
6
10
15
21
28
36
36
```

Python Lambda

Python Lambda

- A lambda function is an anonymous function that is tiny in size.

- A lambda function can have an unlimited number of arguments but only one expression.

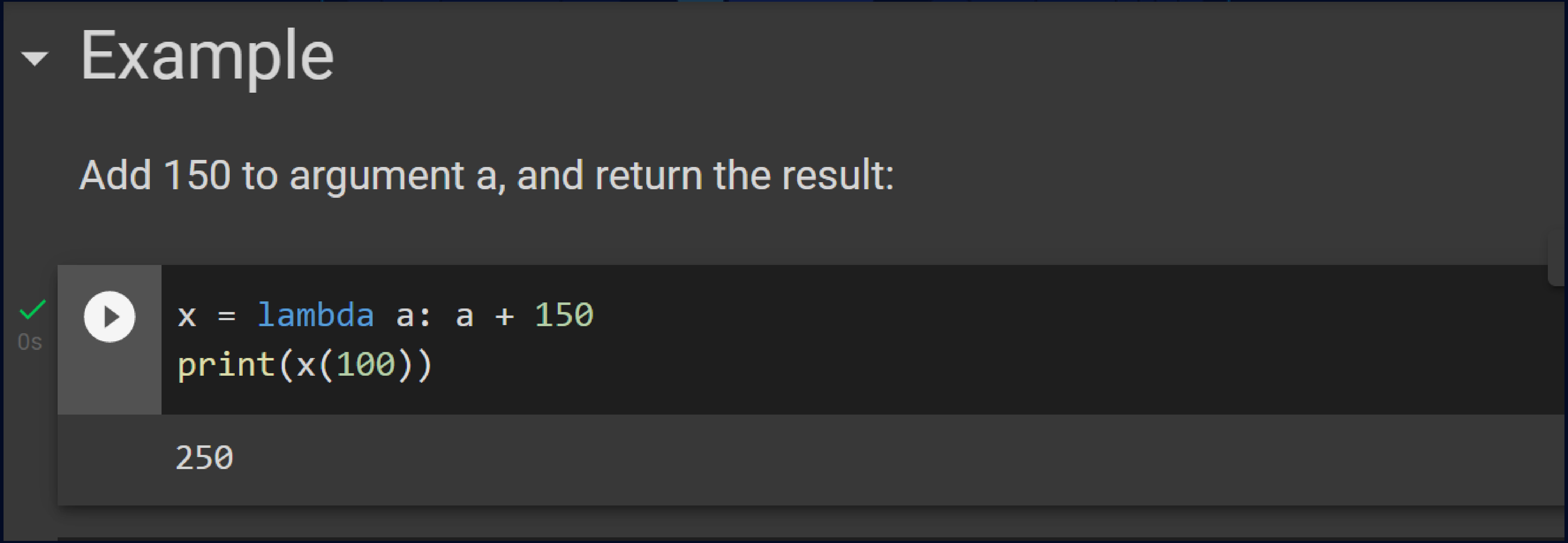


Syntax

```
lambda arguments : expression
```

Python Lambda

Syntax



Example

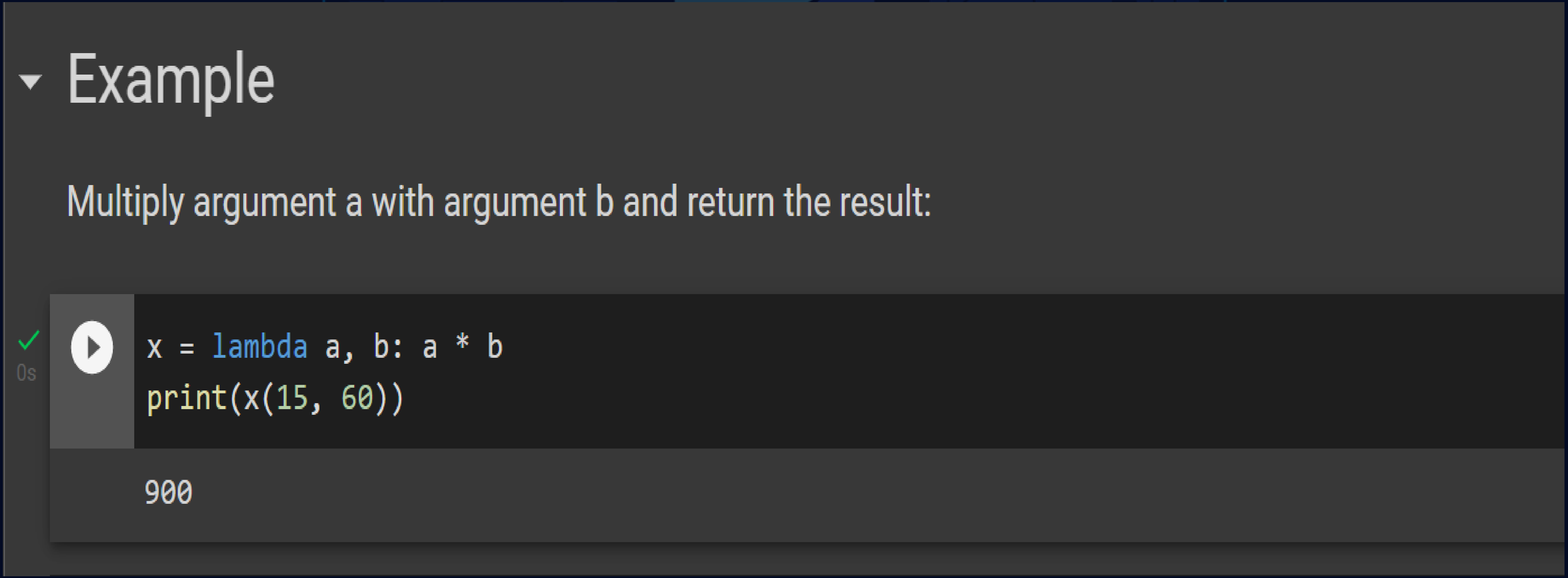
Add 150 to argument a, and return the result:

```
x = lambda a: a + 150
print(x(100))
```

```
250
```

Python Lambda

Lambda functions can accept an unlimited number of arguments:



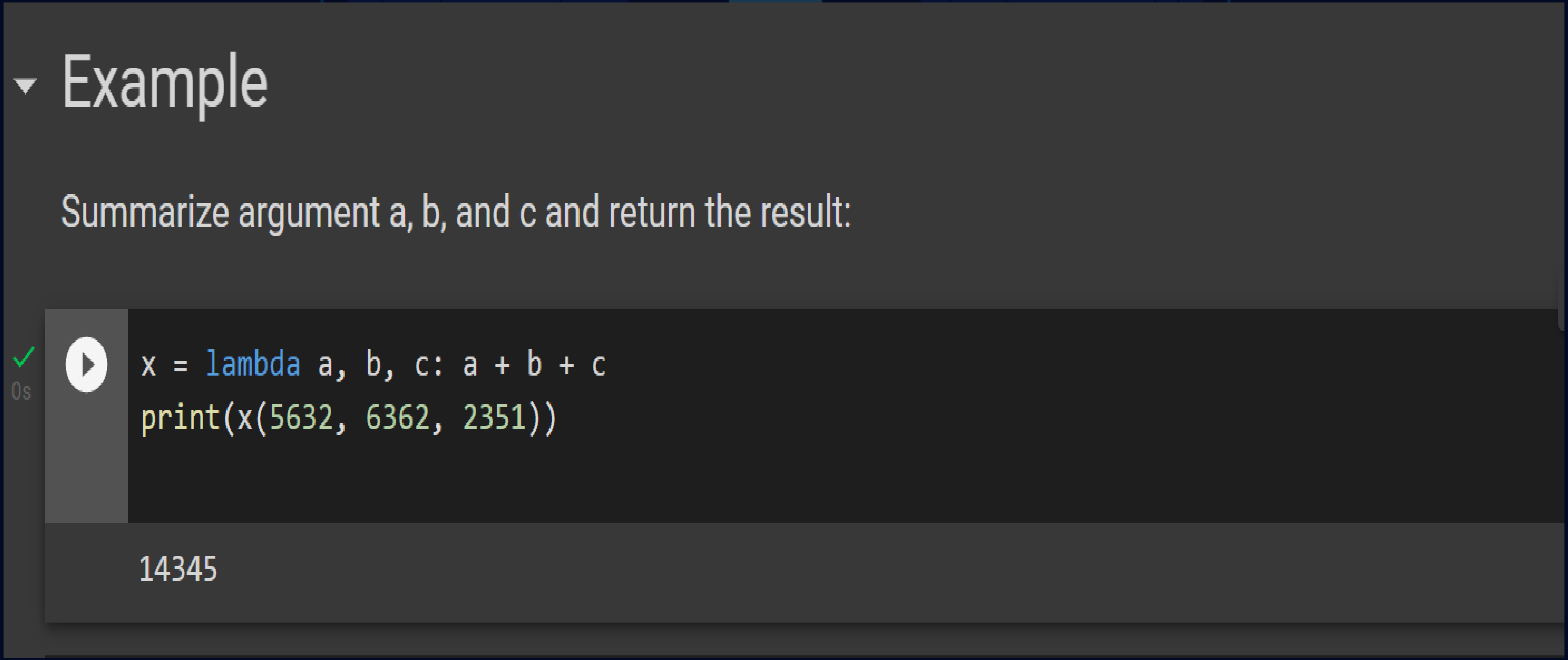
Example

Multiply argument a with argument b and return the result:

```
x = lambda a, b: a * b
print(x(15, 60))
```

```
900
```

Python Lambda



Example

Summarize argument a, b, and c and return the result:

```
x = lambda a, b, c: a + b + c
print(x(5632, 6362, 2351))
```

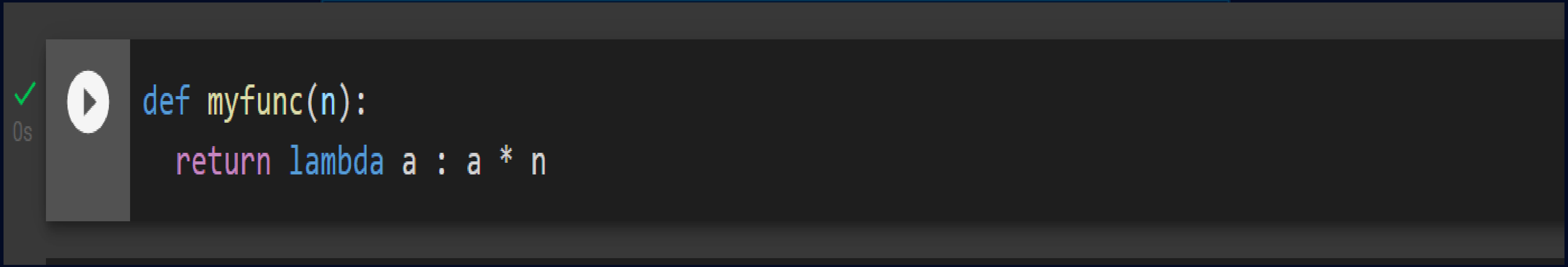
```
14345
```

Python Lambda

Why Use Lambda Functions?

The power of lambda is best demonstrated when used as an anonymous function inside another function.

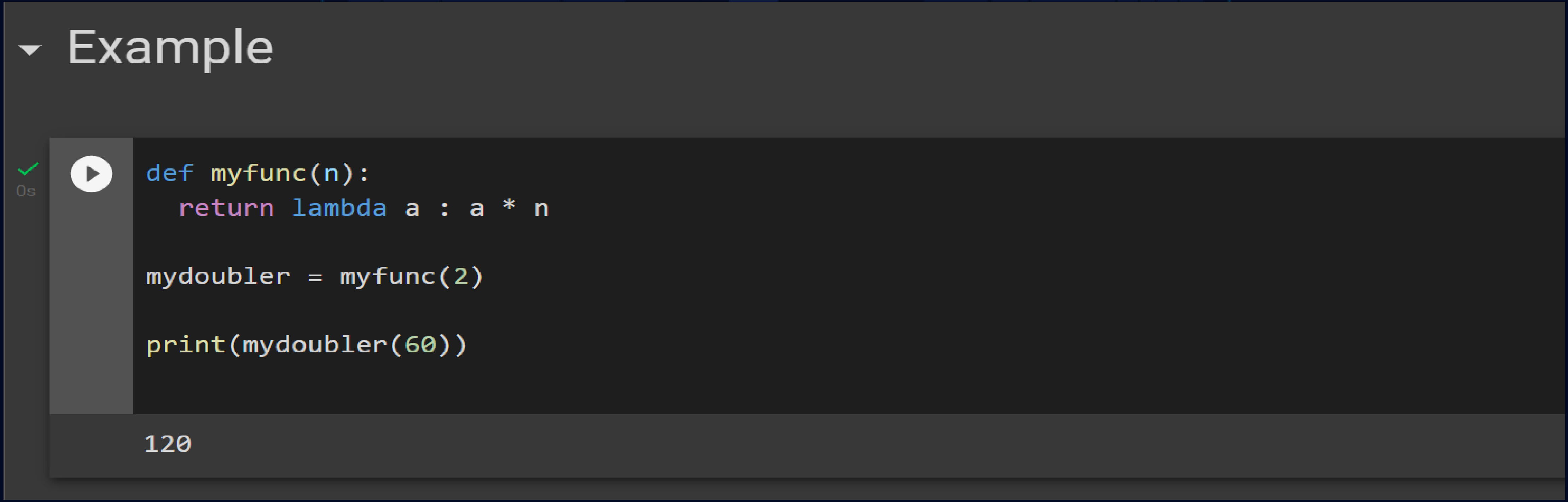
Assume you have a function definition that takes one argument and multiplies it by an unknown number:



```python
def myfunc(n):
  return lambda a : a * n
```

Python Lambda

Make a function using that function definition that always doubles the number you pass in:



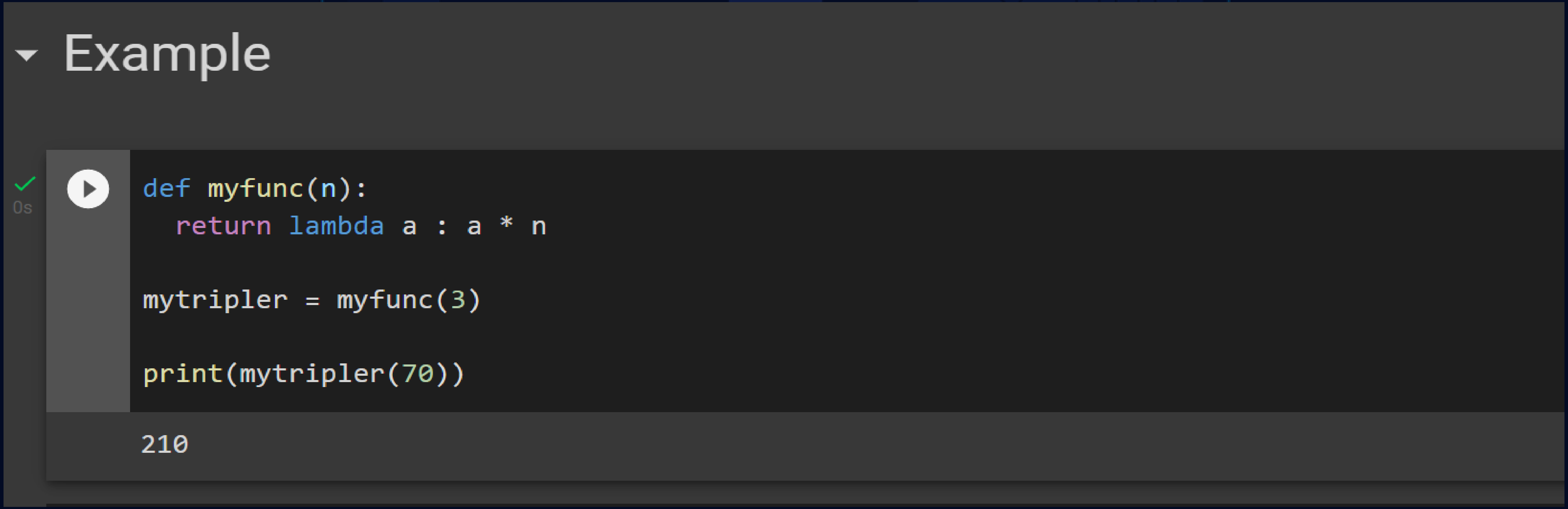
Example

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)

print(mydoubler(60))
```

```
120
```

Python Lambda

Alternatively, you can use the same function definition to create a function that always triples the number you pass in:



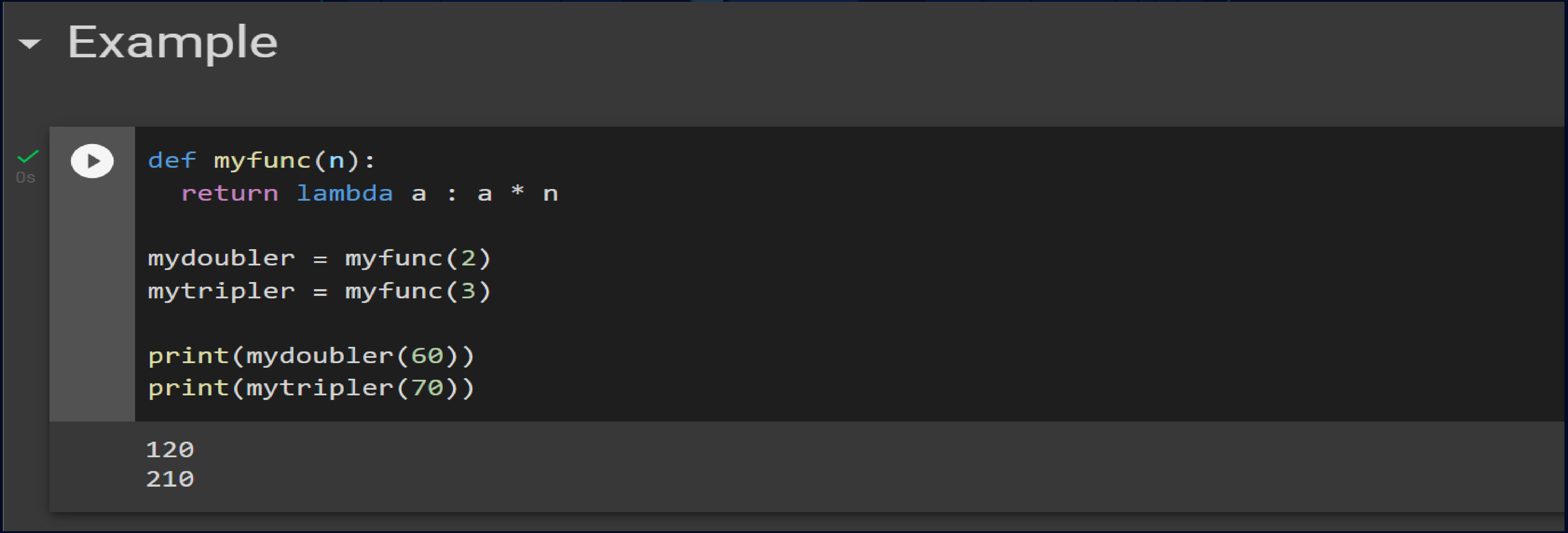
Example

```
def myfunc(n):
  return lambda a : a * n

mytripler = myfunc(3)

print(mytripler(70))
```

```
210
```

Python Lambda

Alternatively, in the same program, use the same function definition to create both functions:

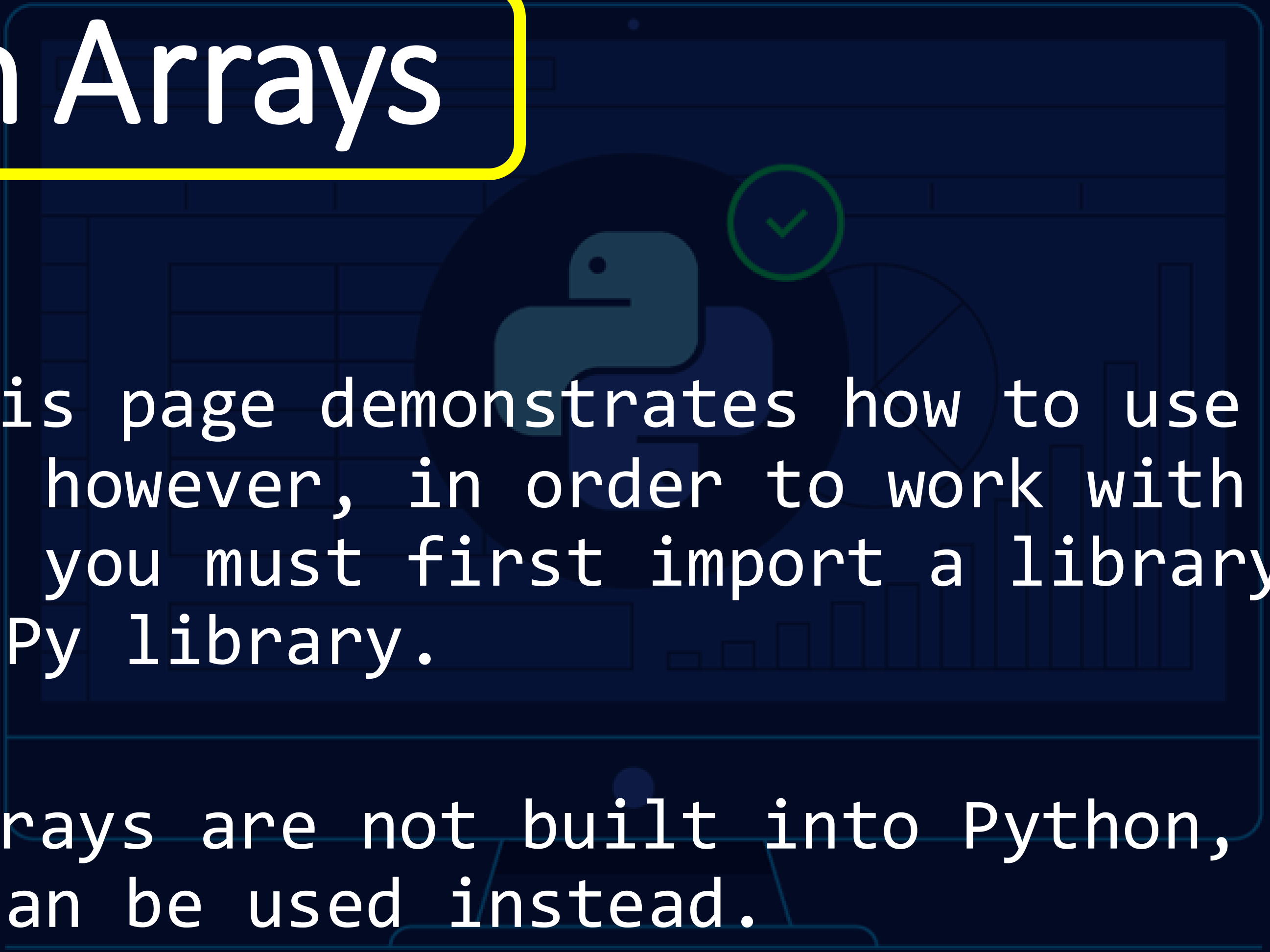


```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(60))
print(mytripler(70))
```

```
120
210
```

Use lambda functions when an anonymous function is required for a short period of time.

Python Arrays



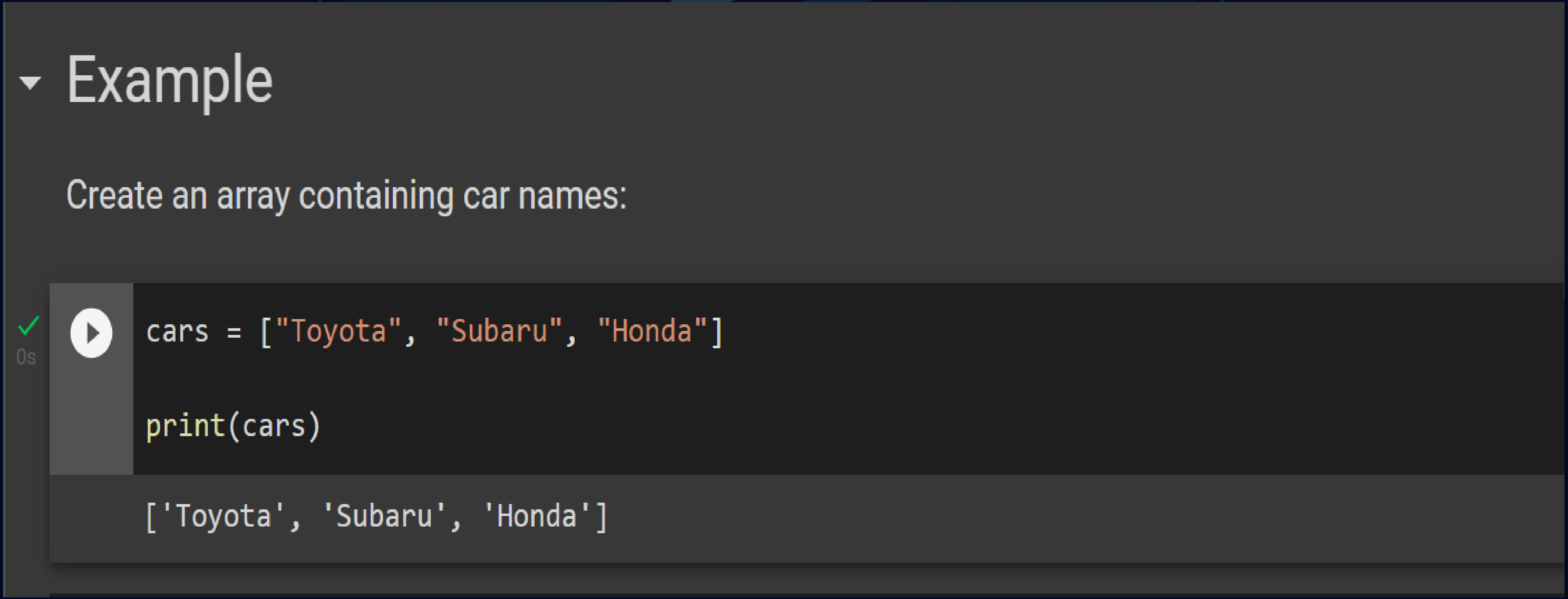
Arrays

Note: This page demonstrates how to use LISTS as ARRAYS; however, in order to work with arrays in Python, you must first import a library, such as the NumPy library.

Note: Arrays are not built into Python, but Python Lists can be used instead.

Python Arrays

Arrays are used to hold a collection of values in a single variable:



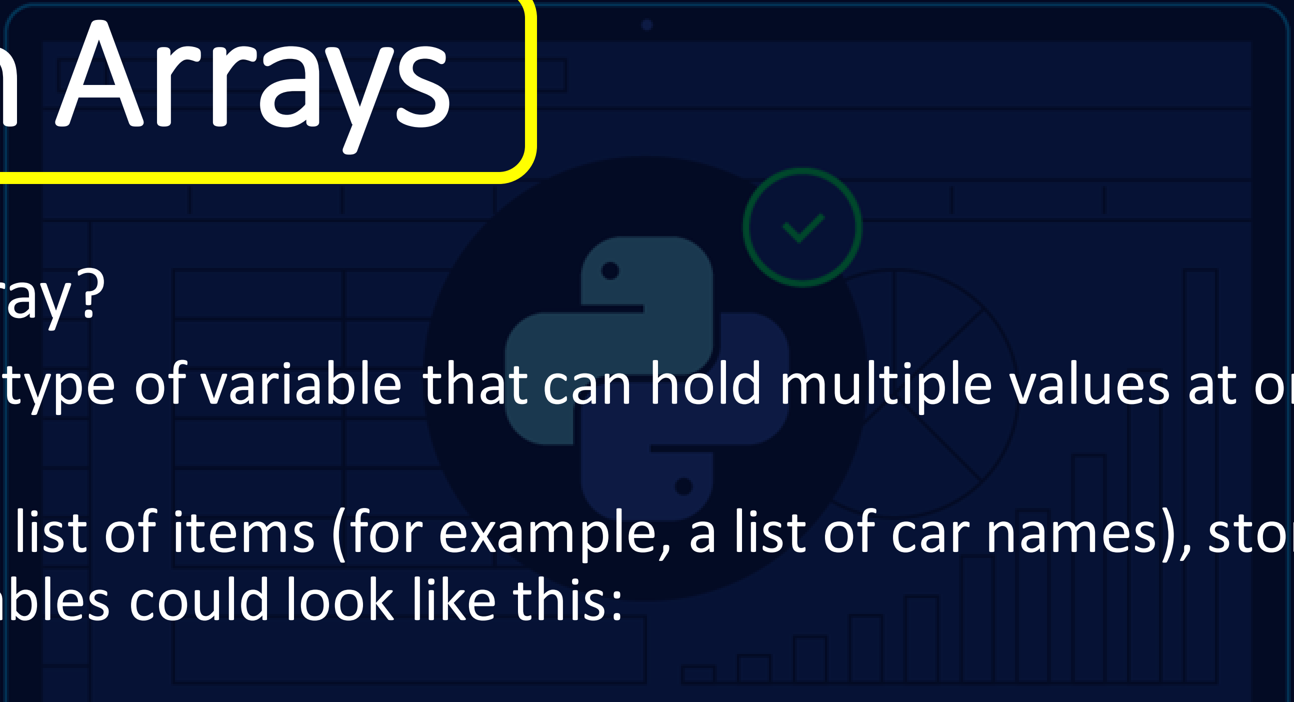
Example

Create an array containing car names:

```
cars = ["Toyota", "Subaru", "Honda"]

print(cars)
```

```
['Toyota', 'Subaru', 'Honda']
```

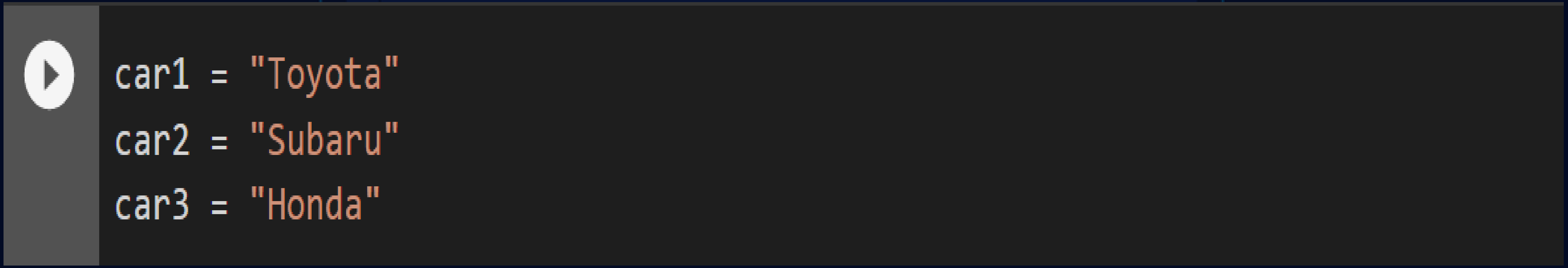
Python Arrays



What is an Array?

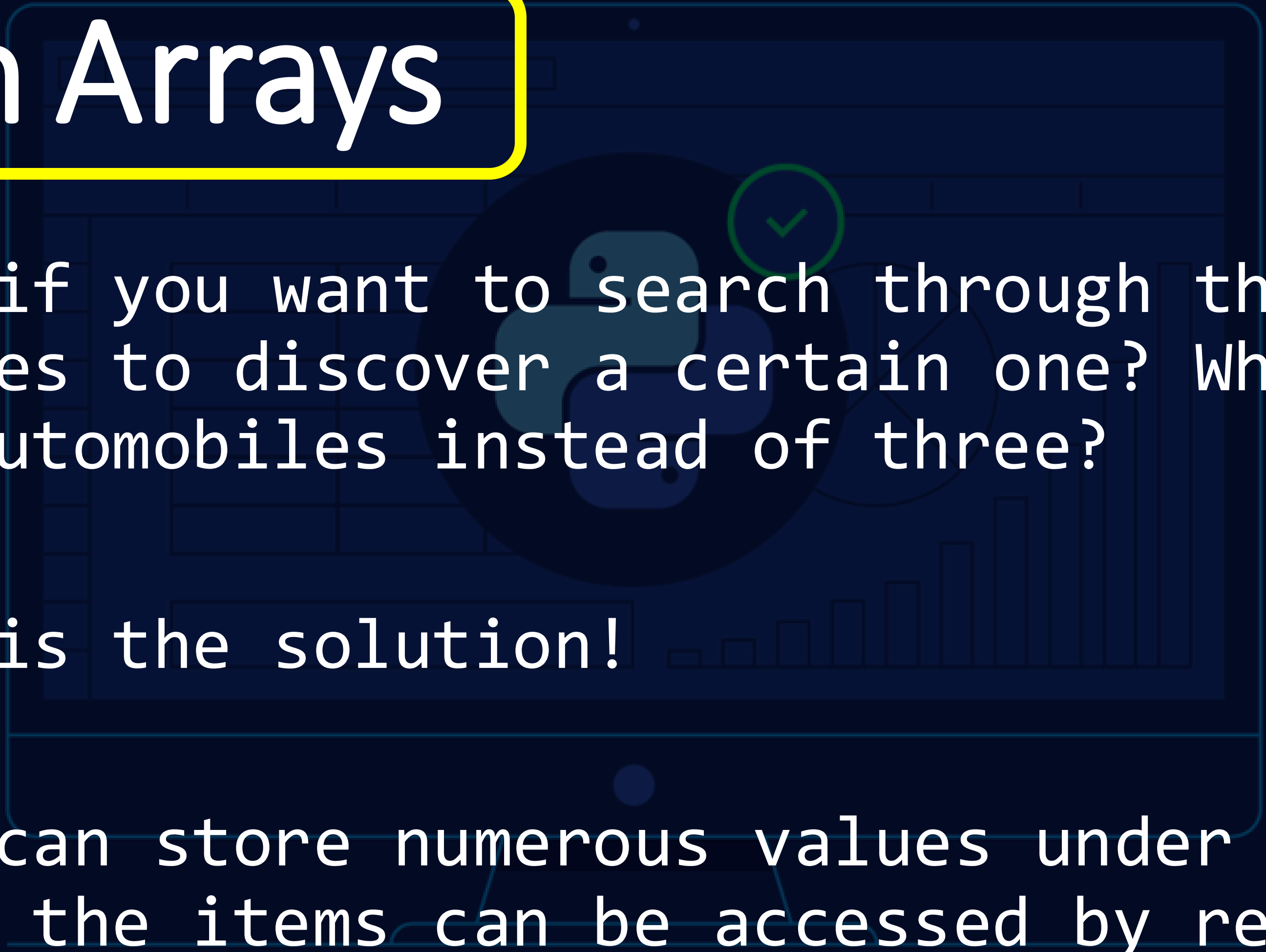
An array is a type of variable that can hold multiple values at once.

If you have a list of items (for example, a list of car names), storing the cars in single variables could look like this:



```
car1 = "Toyota"
car2 = "Subaru"
car3 = "Honda"
```

Python Arrays



- But what if you want to search through the automobiles to discover a certain one? What if you had 300 automobiles instead of three?

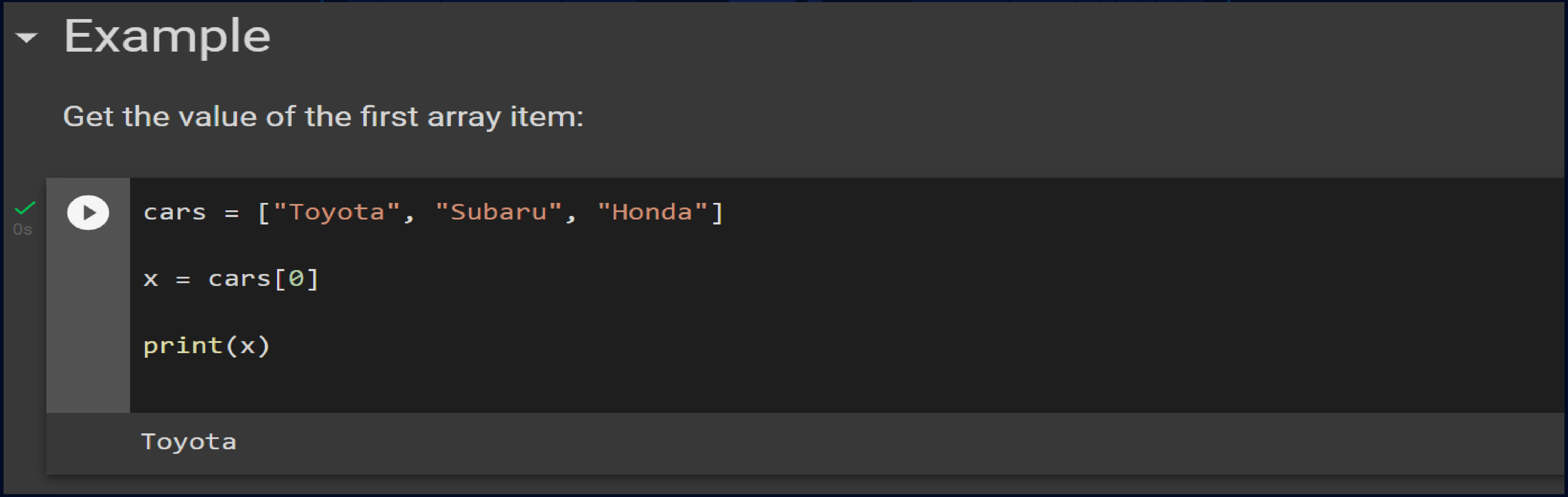
- An array is the solution!

- An array can store numerous values under a single name, and the items can be accessed by referring to an index number.

Python Arrays

Access the Elements of an Array

The index number is used to refer to an array element.



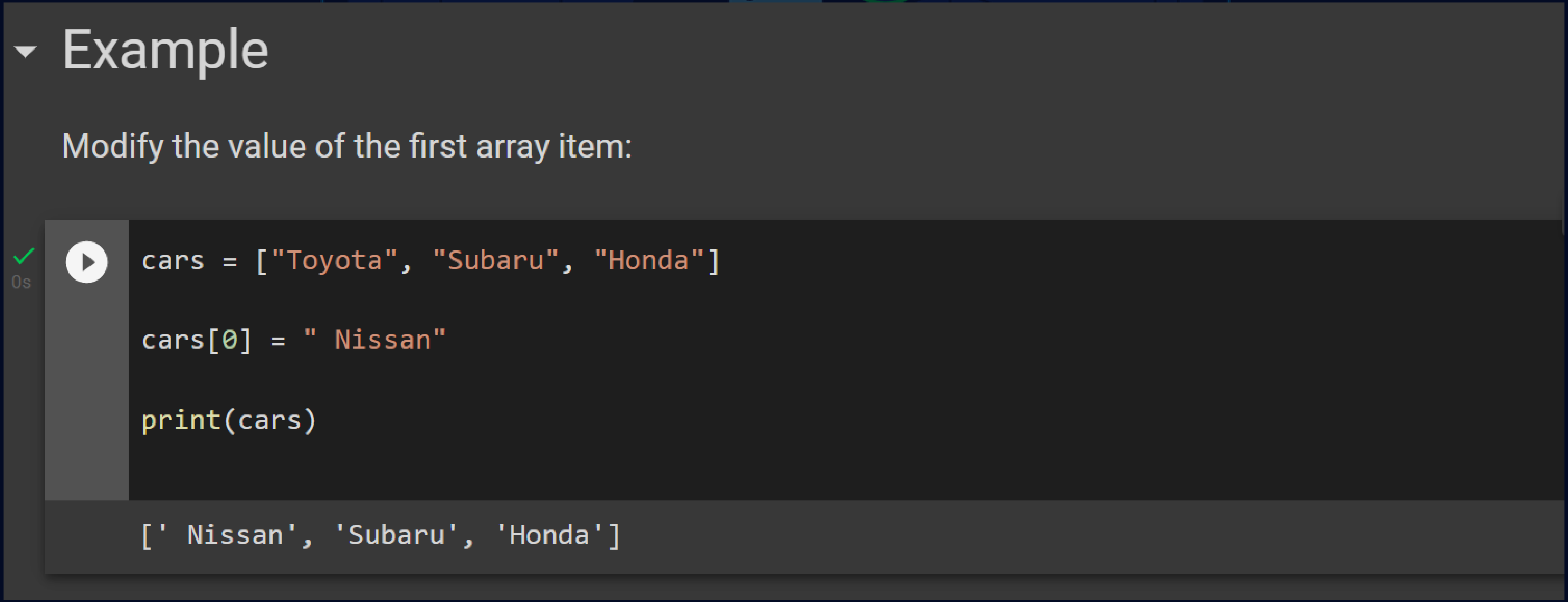
Example

Get the value of the first array item:

```
cars = ["Toyota", "Subaru", "Honda"]

x = cars[0]

print(x)
```

```
Toyota
```

Python Arrays



Example

Modify the value of the first array item:

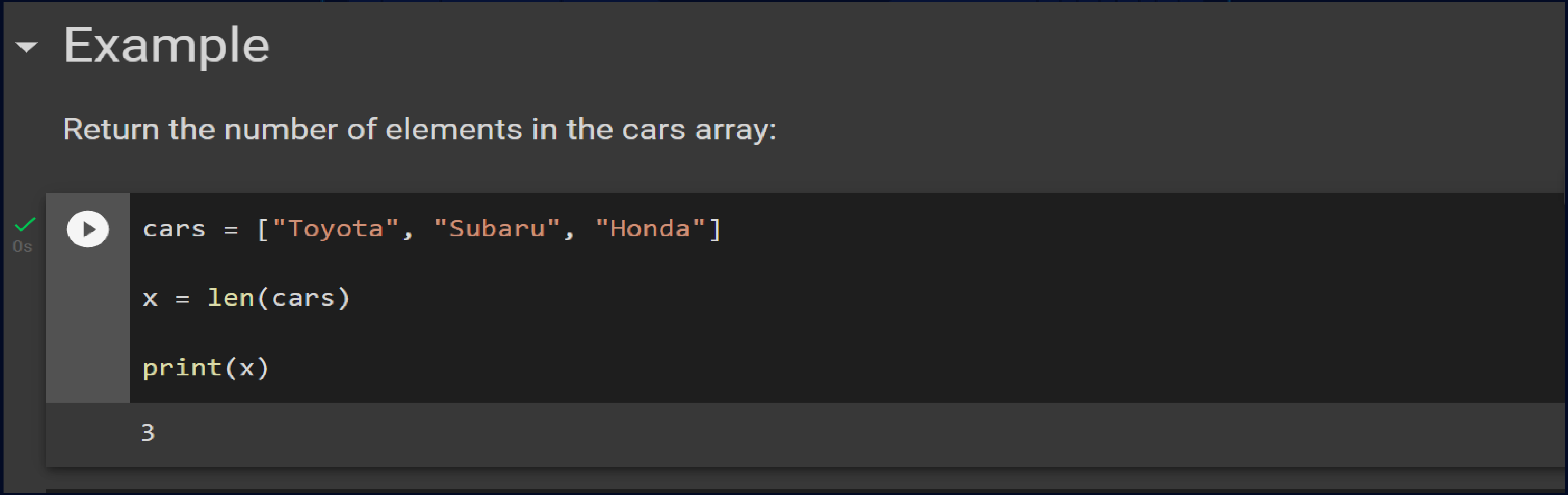
```
cars = ["Toyota", "Subaru", "Honda"]

cars[0] = " Nissan"

print(cars)
```

```
[' Nissan', 'Subaru', 'Honda']
```

Python Arrays

The Length of an Array

To determine the length of an array, use the len() function (the number of elements in an array).



Example

Return the number of elements in the cars array:

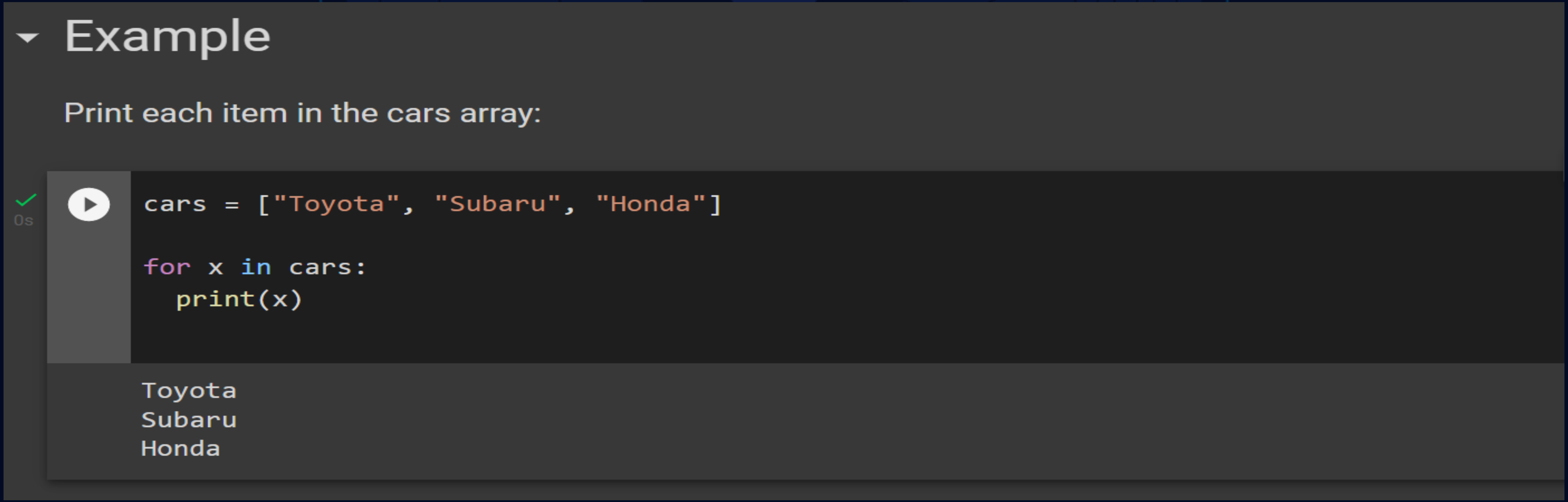
```
cars = ["Toyota", "Subaru", "Honda"]

x = len(cars)

print(x)
```

```
3
```

Python Arrays

Looping Array Elements

The for in loop can be used to loop through all the elements of an array.



Example

Print each item in the cars array:

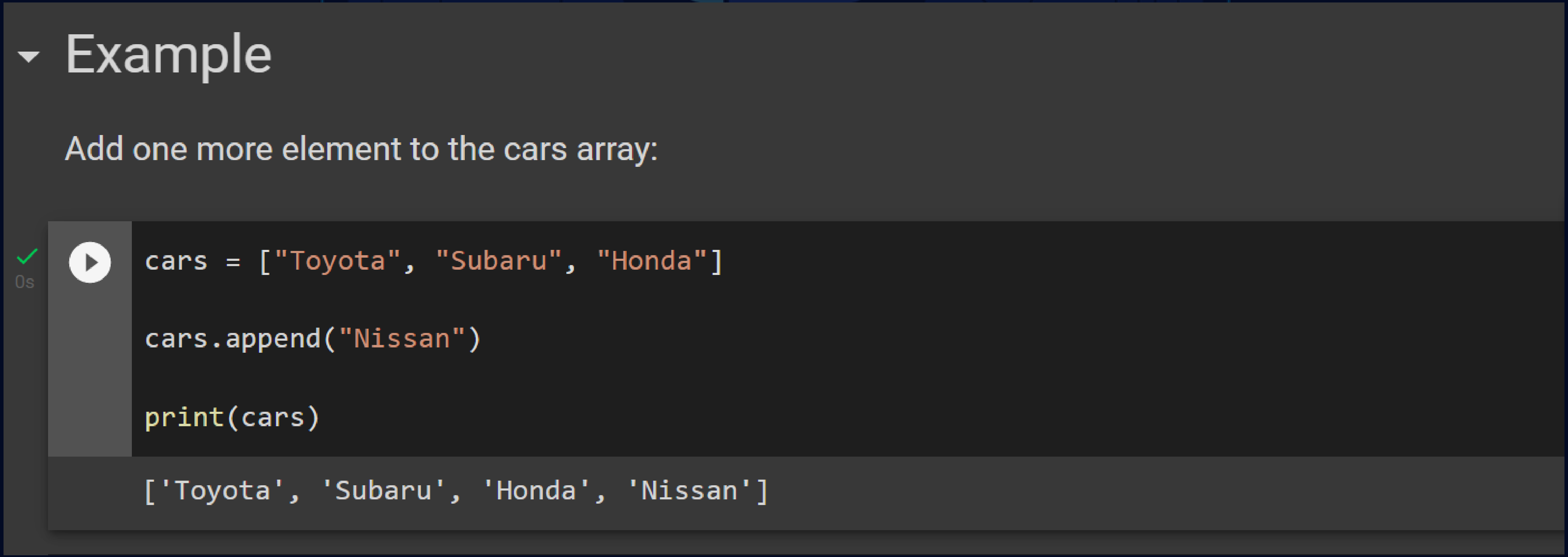
```
cars = ["Toyota", "Subaru", "Honda"]

for x in cars:
  print(x)
```

```
Toyota
Subaru
Honda
```

Python Arrays

Adding Array Elements

To add an element to an array, use the append() method.



Example

Add one more element to the cars array:

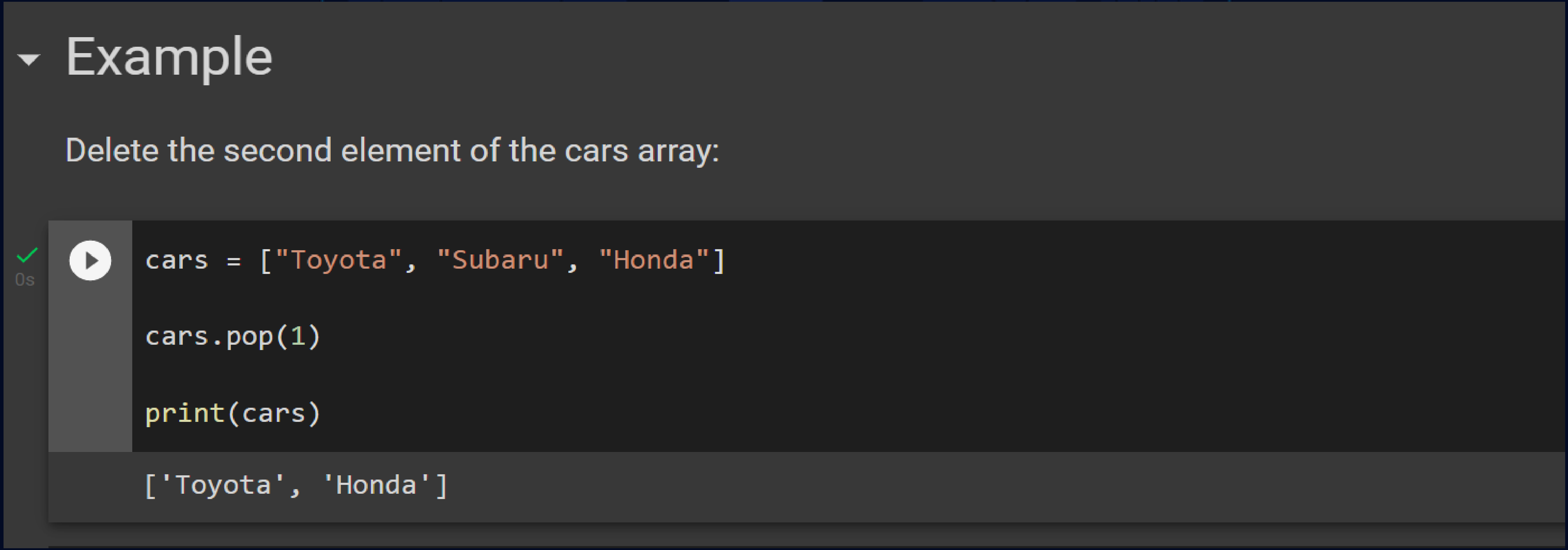
```
cars = ["Toyota", "Subaru", "Honda"]

cars.append("Nissan")

print(cars)
```

```
['Toyota', 'Subaru', 'Honda', 'Nissan']
```

Python Arrays

Removing Array Elements

To remove an element from an array, use the pop() method.



Example

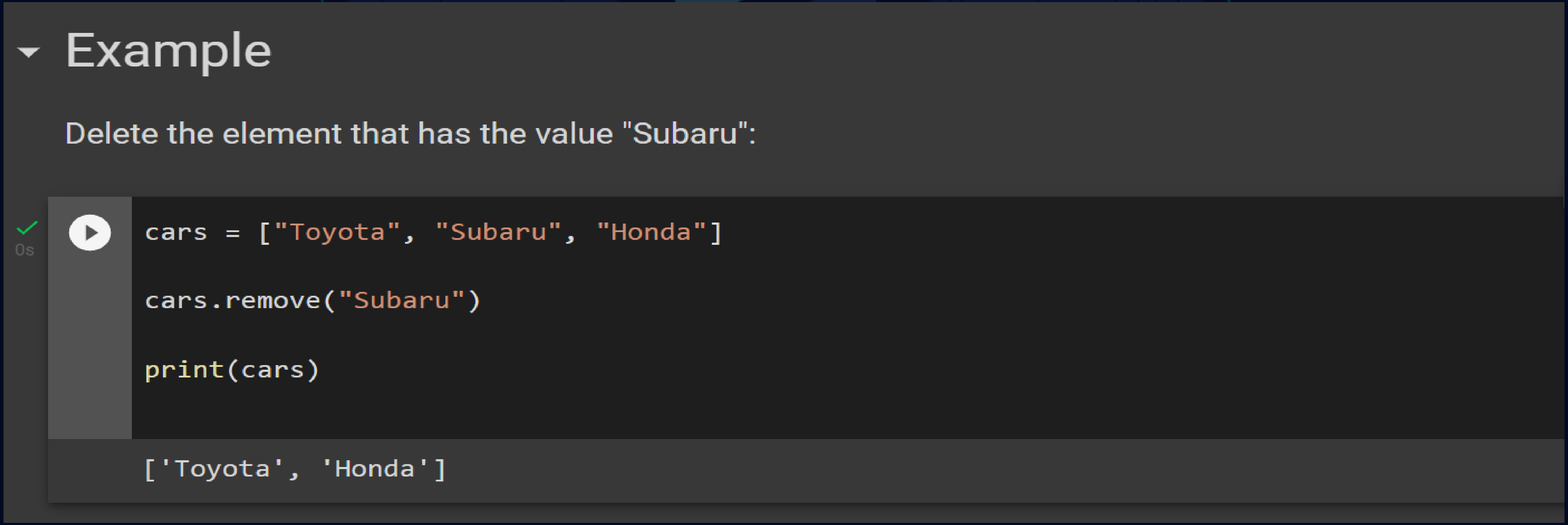
Delete the second element of the cars array:

```
cars = ["Toyota", "Subaru", "Honda"]

cars.pop(1)

print(cars)
```

```
['Toyota', 'Honda']
```

Python Arrays

To remove an element from an array, use the remove() method.



Example

Delete the element that has the value "Subaru":

```
cars = ["Toyota", "Subaru", "Honda"]

cars.remove("Subaru")

print(cars)
```

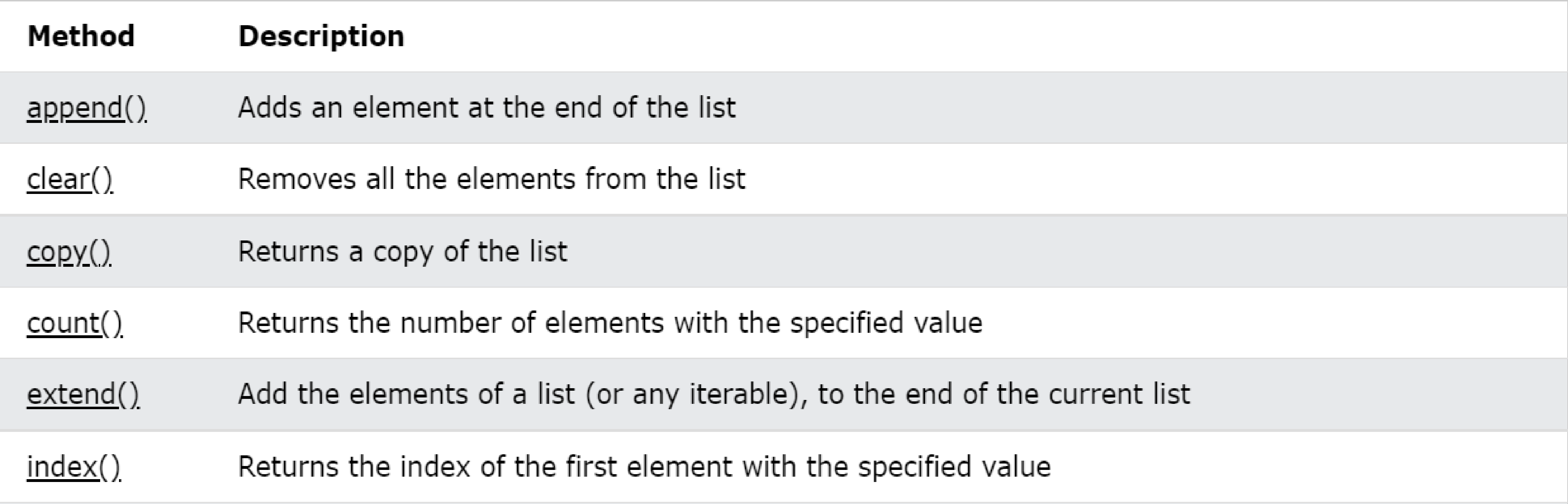
```
['Toyota', 'Honda']
```

Note: The list's remove() method only removes the first occurrence of the specified value.

Python Arrays

Array Methods

Python has a set of built-in methods that you can use on lists/arrays.

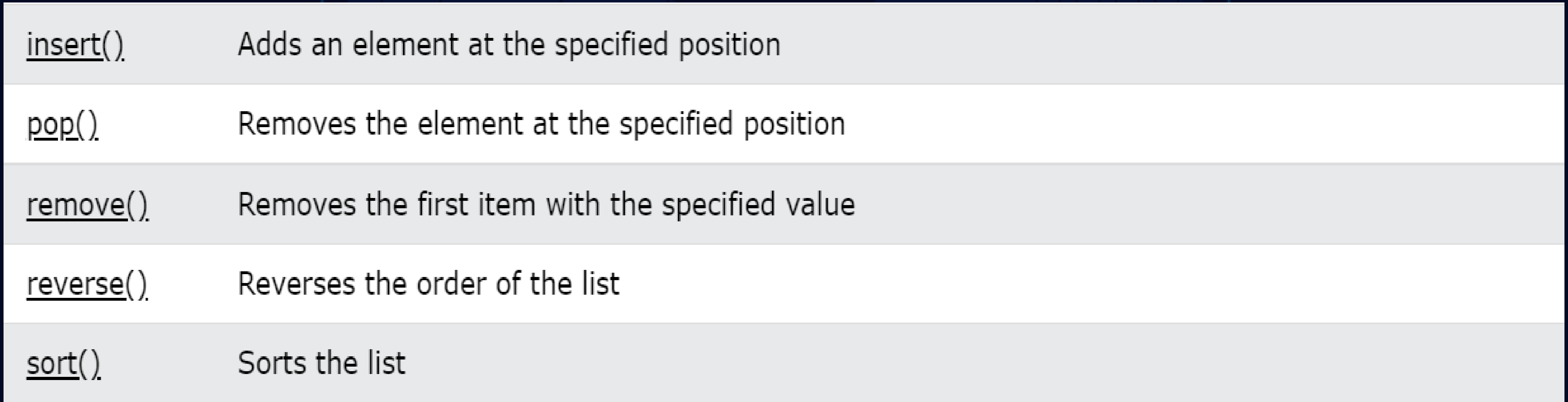


Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value

Python Arrays

Array Methods

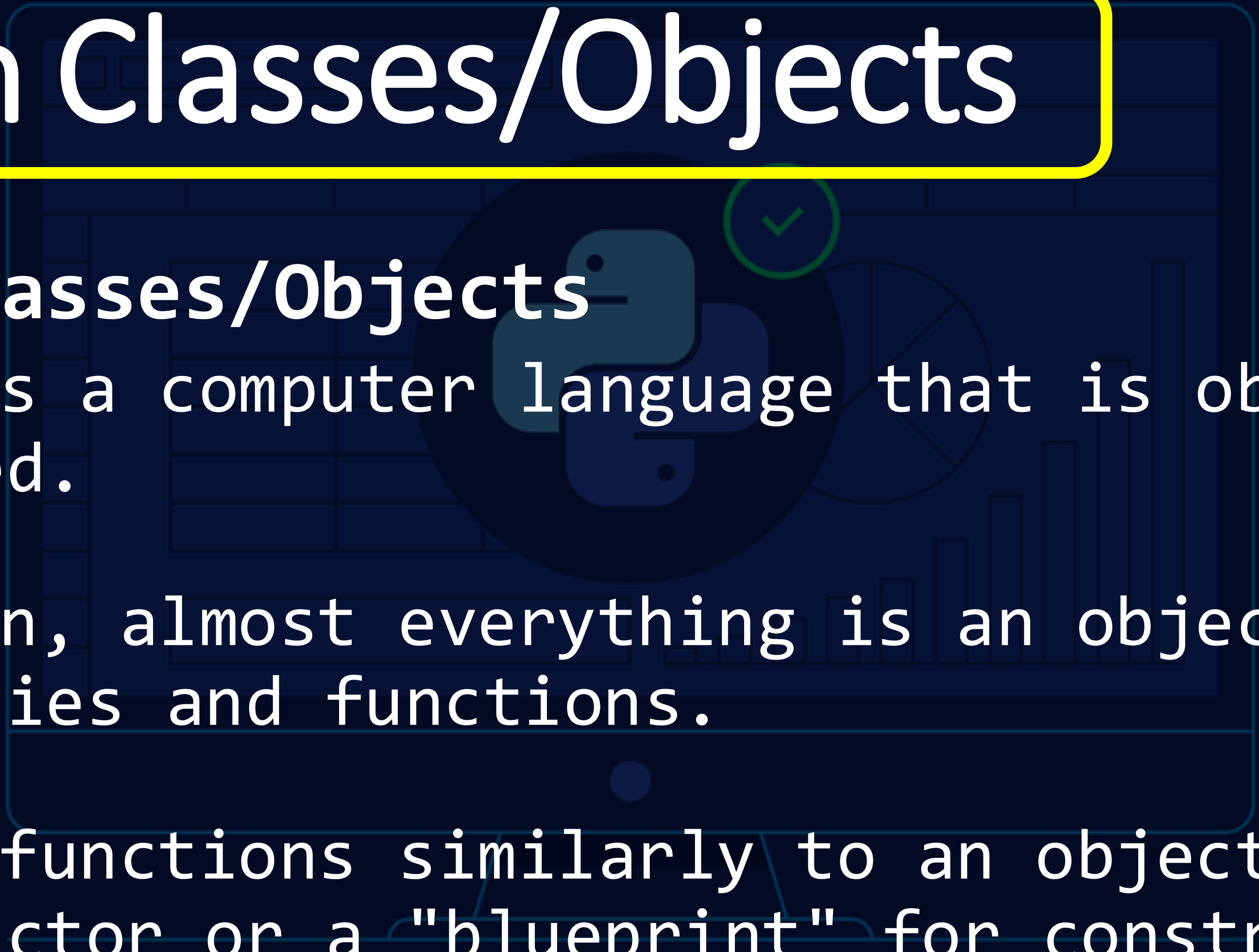
Python has a set of built-in methods that you can use on lists/arrays.



insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Note: Python does not have built-in support for Arrays, but Python Lists can be used instead.

Python Classes/Objects



Python Classes/Objects

Python is a computer language that is object oriented.

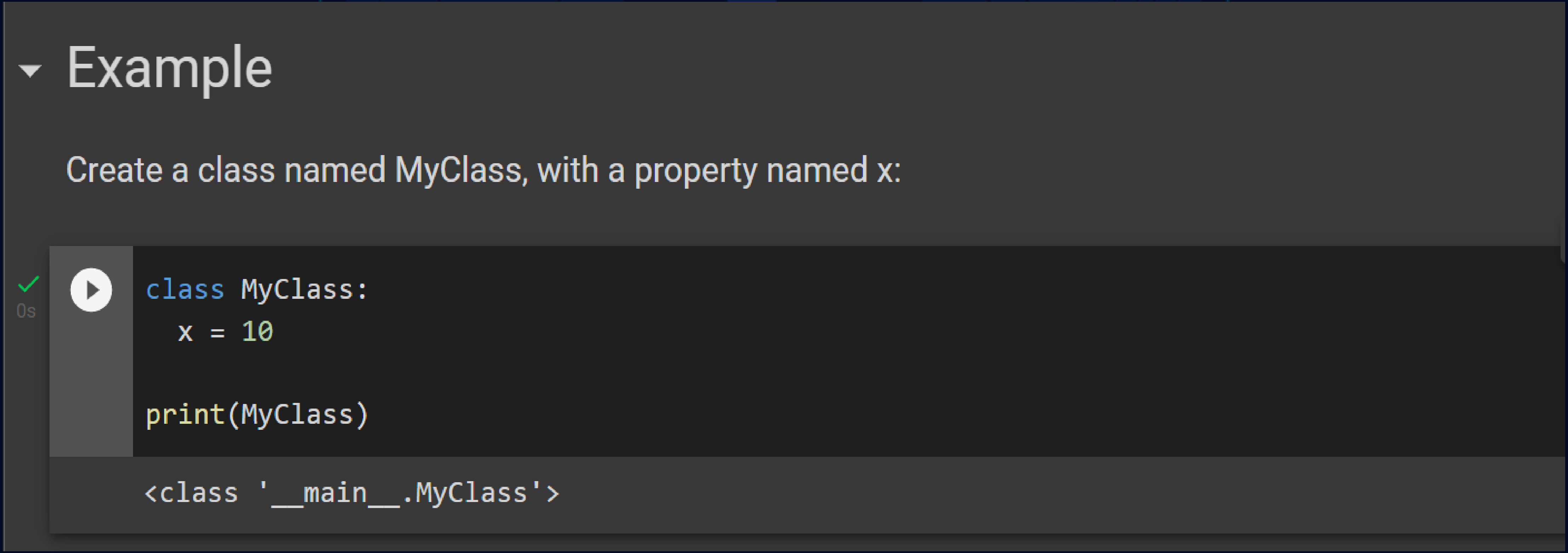
In Python, almost everything is an object with properties and functions.

A Class functions similarly to an object constructor or a "blueprint" for constructing things.

Python Classes/Objects

Create a Class

Use the keyword class to create a class:



Example

Create a class named MyClass, with a property named x:

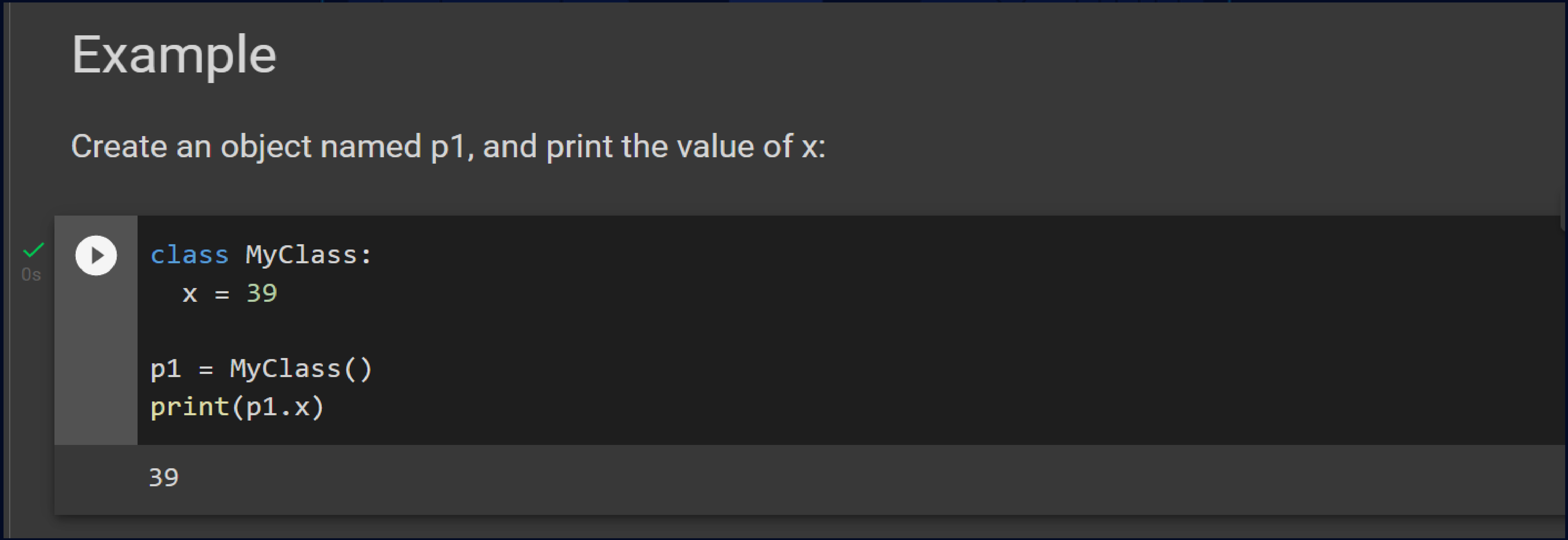
```
class MyClass:
  x = 10

print(MyClass)
```

```
<class '__main__.MyClass'>
```

Python Classes/Objects

Create Object

We can now construct objects using the MyClass class:



Example

Create an object named p1, and print the value of x:

```python
class MyClass:
  x = 39

p1 = MyClass()
print(p1.x)
```

```
39
```

Python Classes/Objects

The __init__() Function

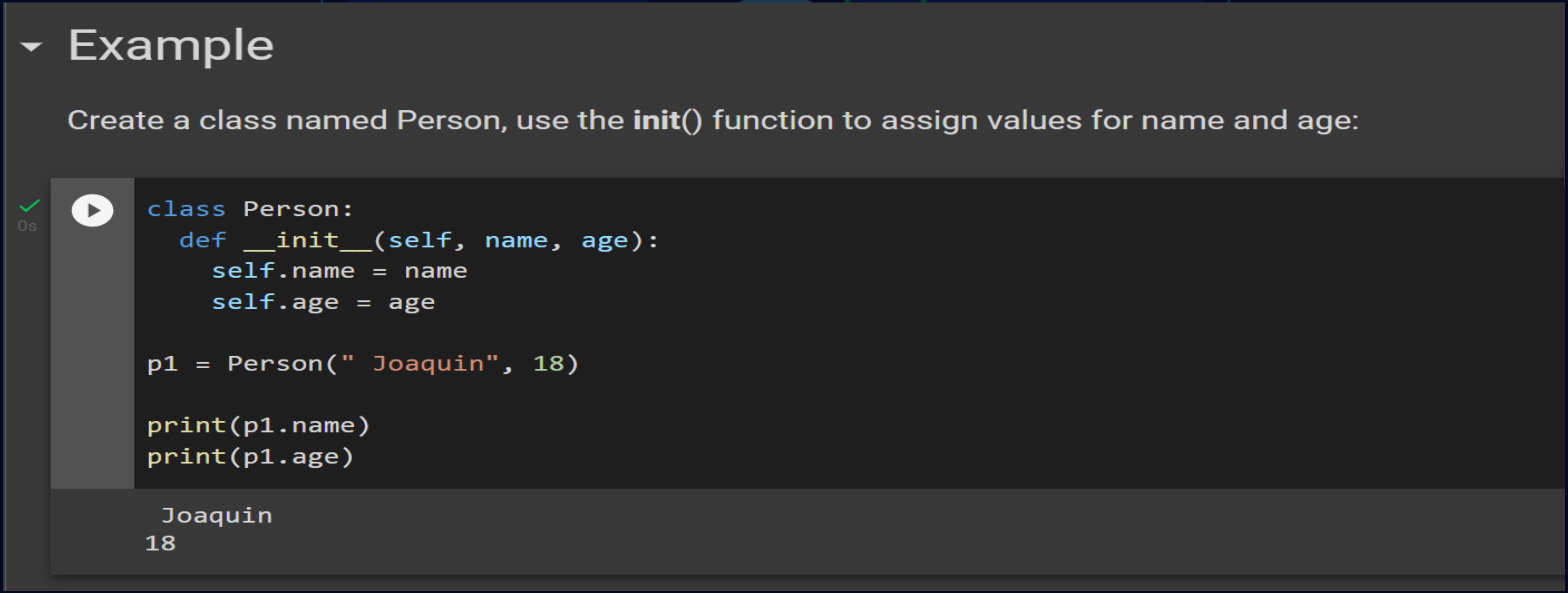
- The examples above are classes and objects in their most basic form, and thus are not particularly useful in real-world applications.

- To comprehend the concept of classes, we must first comprehend the built-in __init__() function.

- Every class has a procedure called __init__() that is always invoked when the class is launched.

- Use the __init__() function to assign values to object properties or to do other activities required when the object is created:

Python Classes/Objects



Example

Create a class named Person, use the **init()** function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person(" Joaquin", 18)

print(p1.name)
print(p1.age)
```

```
 Joaquin
18
```

Note: The __init__() function is called automatically every time the class is being used to create a new object.

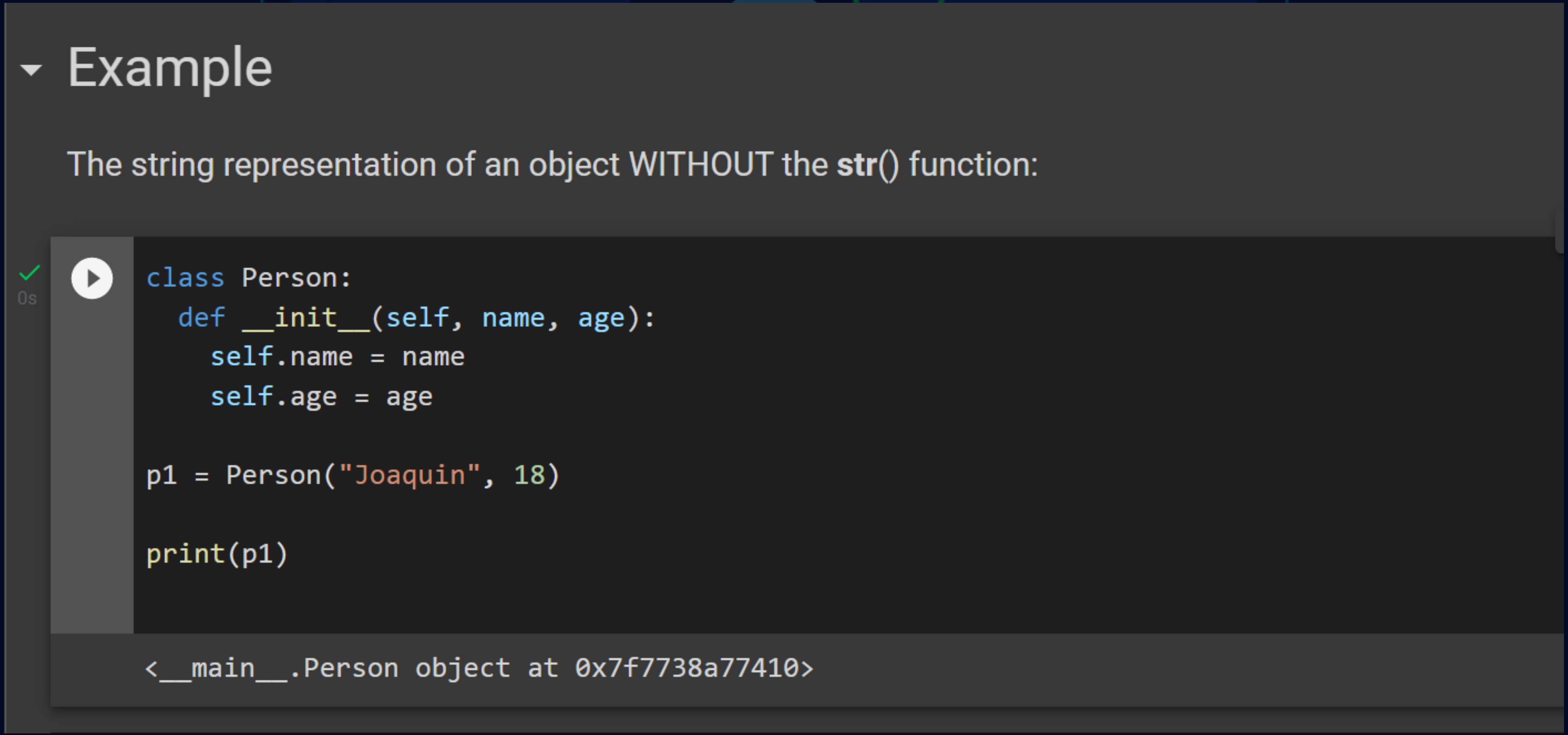
Python Classes/Objects

The __str__() Function

- When a class object is represented as a string, the __str__() function determines what should be returned.

- If the __str__() method is not set, the object's string representation is returned:

Python Classes/Objects



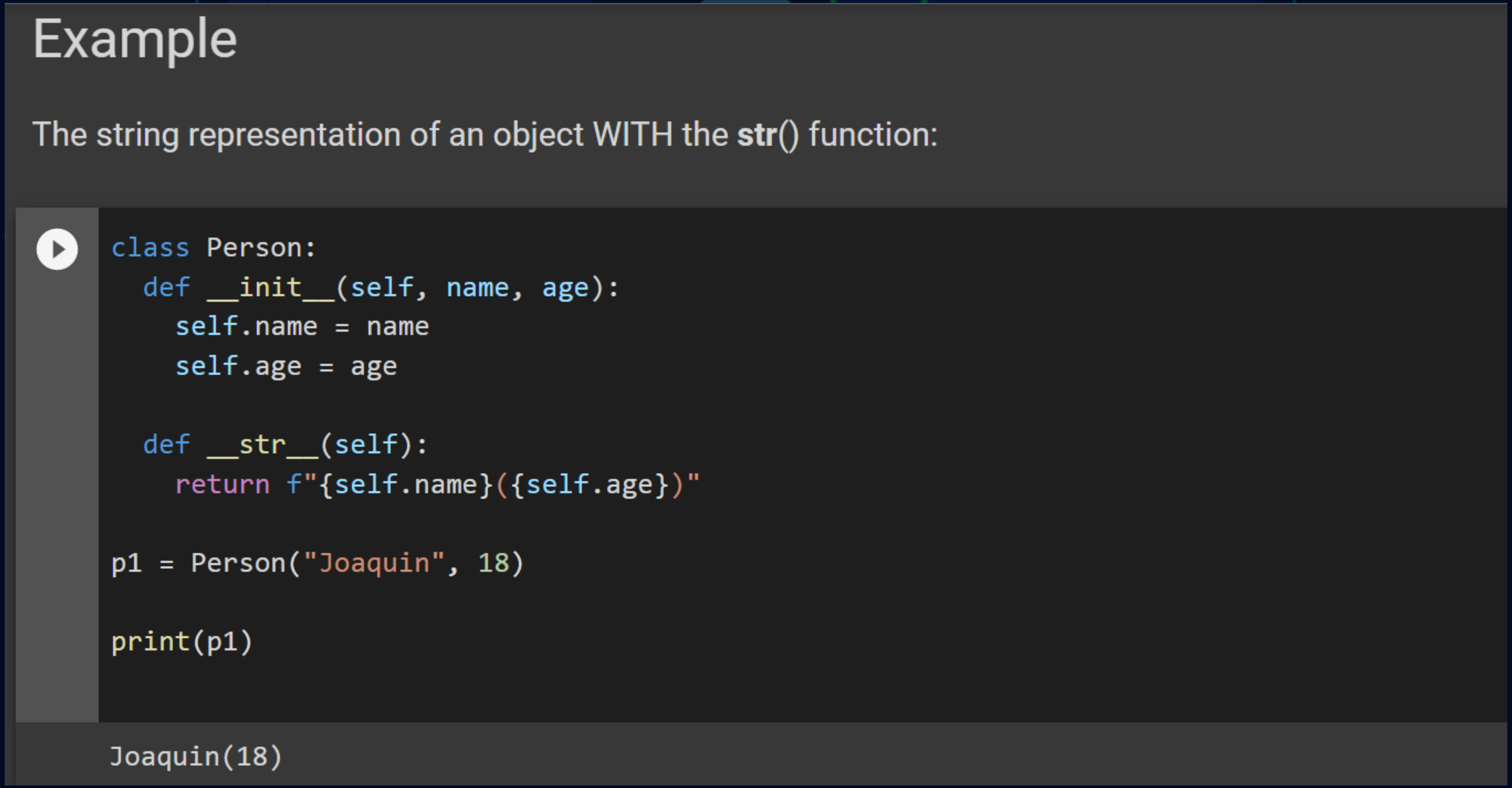
Example

The string representation of an object WITHOUT the **str()** function:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("Joaquin", 18)

print(p1)
```

```
<__main__.Person object at 0x7f7738a77410>
```

Python Classes/Objects



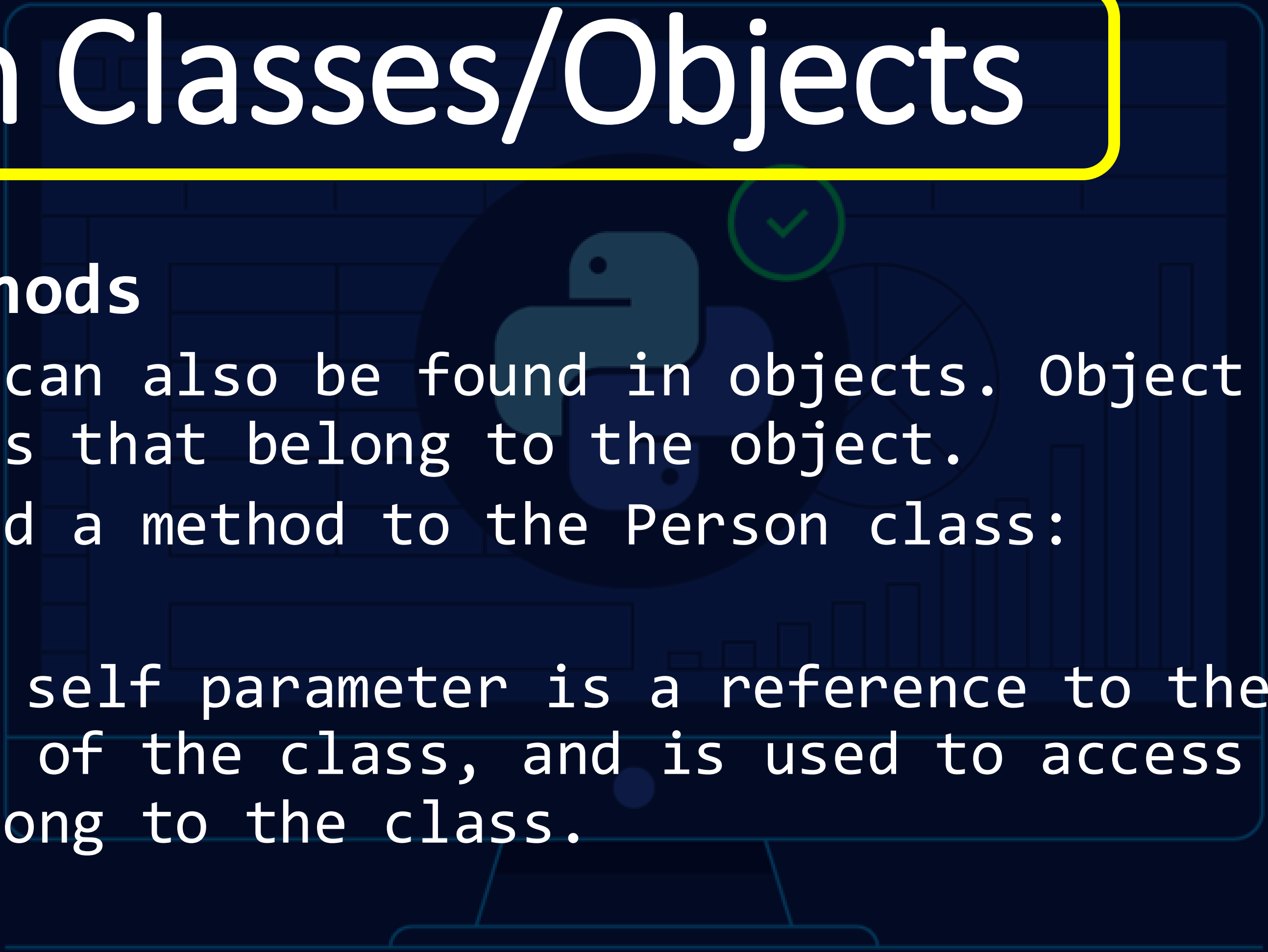
Example

The string representation of an object WITH the **str**() function:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("Joaquin", 18)

print(p1)
```

```
Joaquin(18)
```

Python Classes/Objects



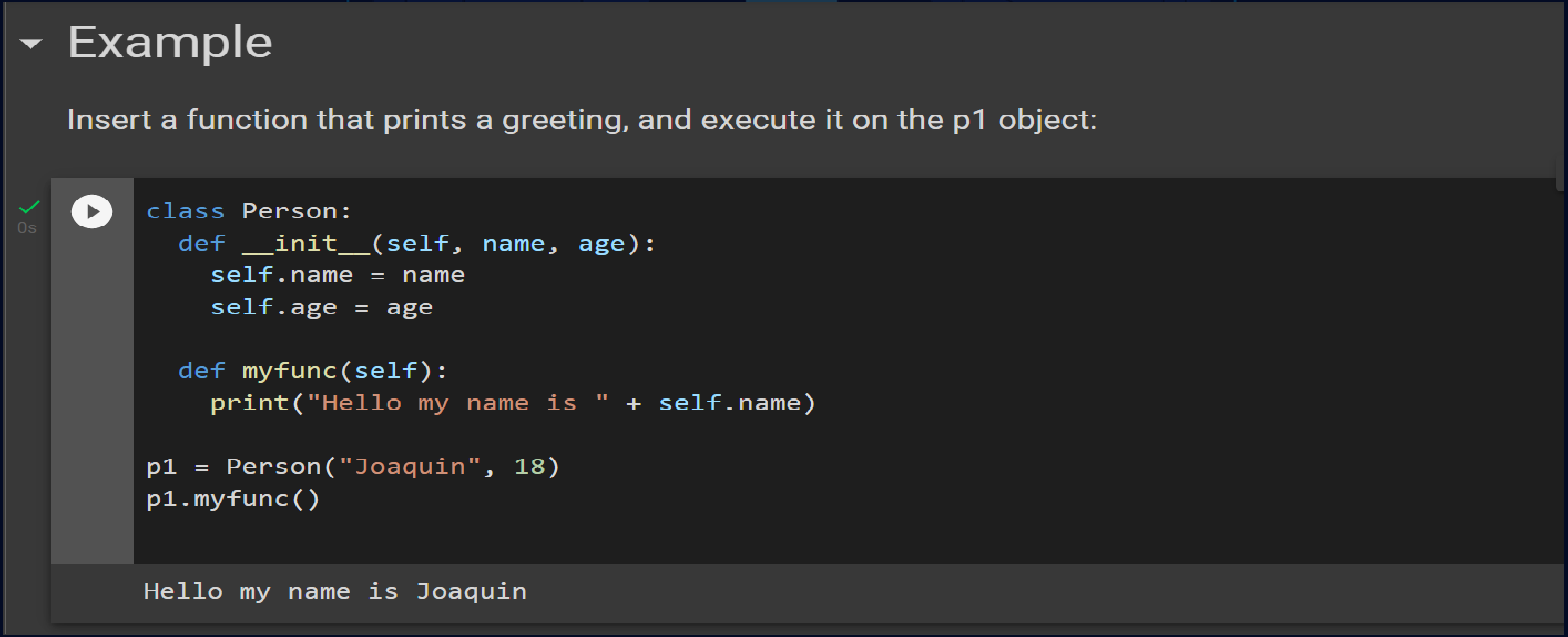
Object Methods

- Methods can also be found in objects. Object methods are functions that belong to the object.
- Let's add a method to the Person class:

Note: The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

EXAMPLE BELOW

Python Classes/Objects



Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("Joaquin", 18)
p1.myfunc()
```

```
Hello my name is Joaquin
```

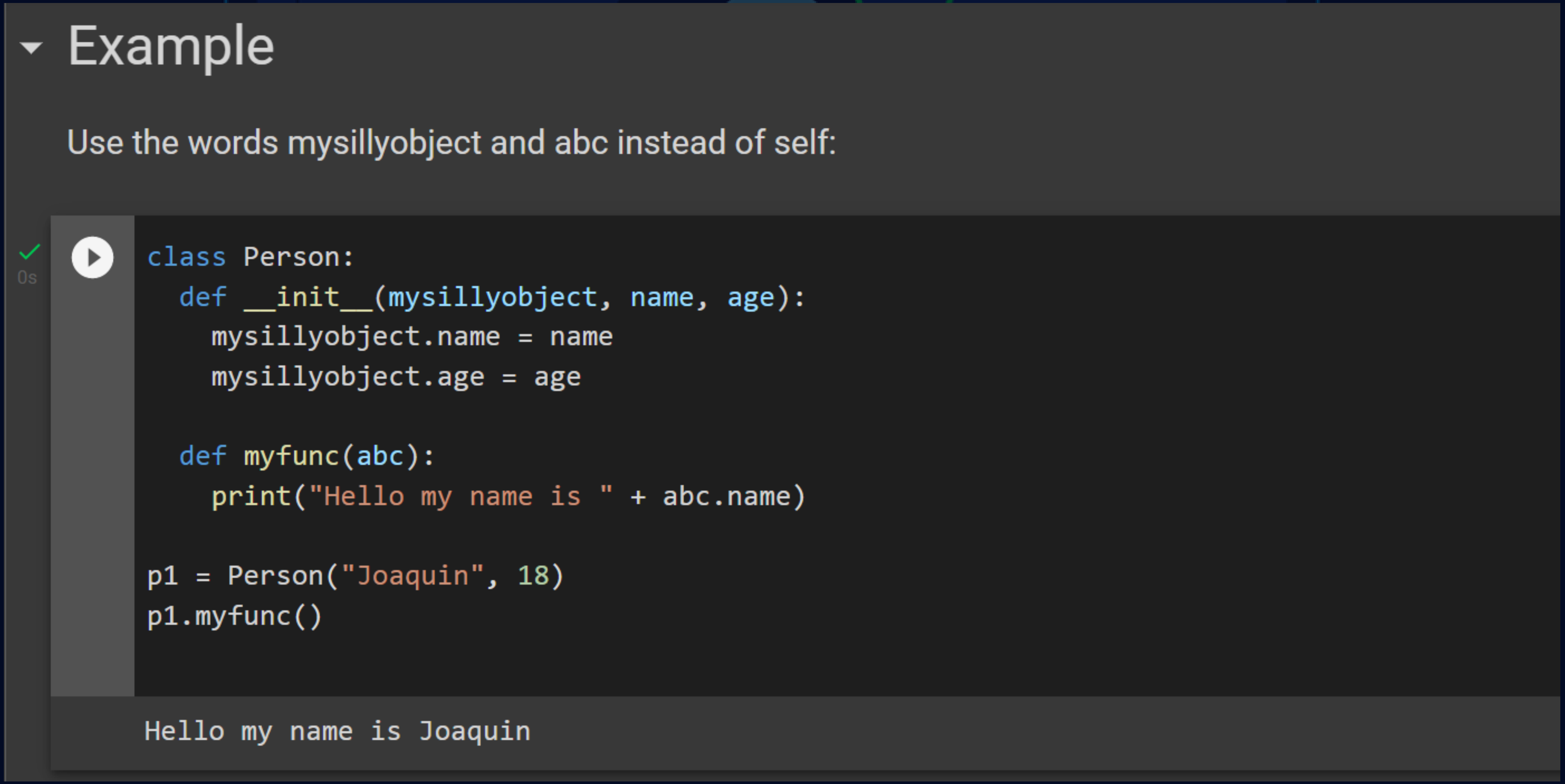
Python Classes/Objects

The self Parameter

- The self parameter is a reference to the current instance of the class and is used to access class variables.
- It does not have to be called self; it can be anything you want, but it must be the first parameter of any function in the class:

EXAMPLE BELOW

Python Classes/Objects



Example

Use the words mysillyobject and abc instead of self:

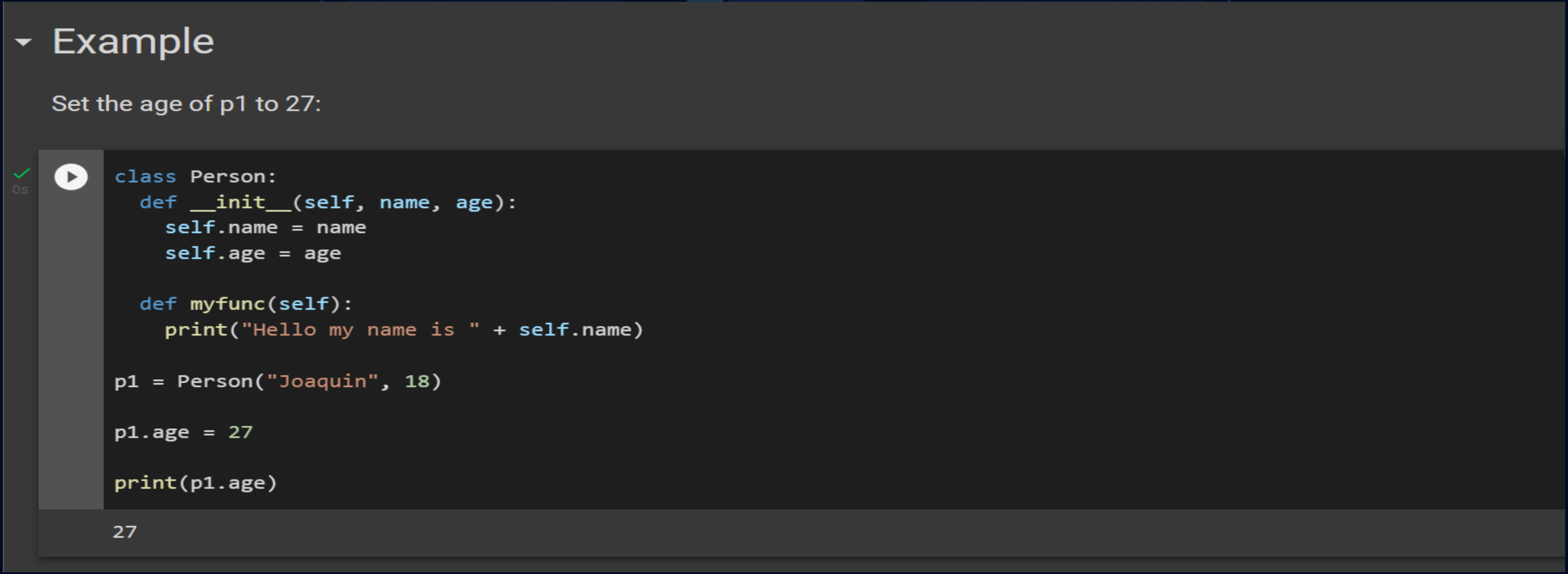
```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("Joaquin", 18)
p1.myfunc()
```

```
Hello my name is Joaquin
```

Python Classes/Objects

Modify Object Properties

You can modify properties on objects like this:

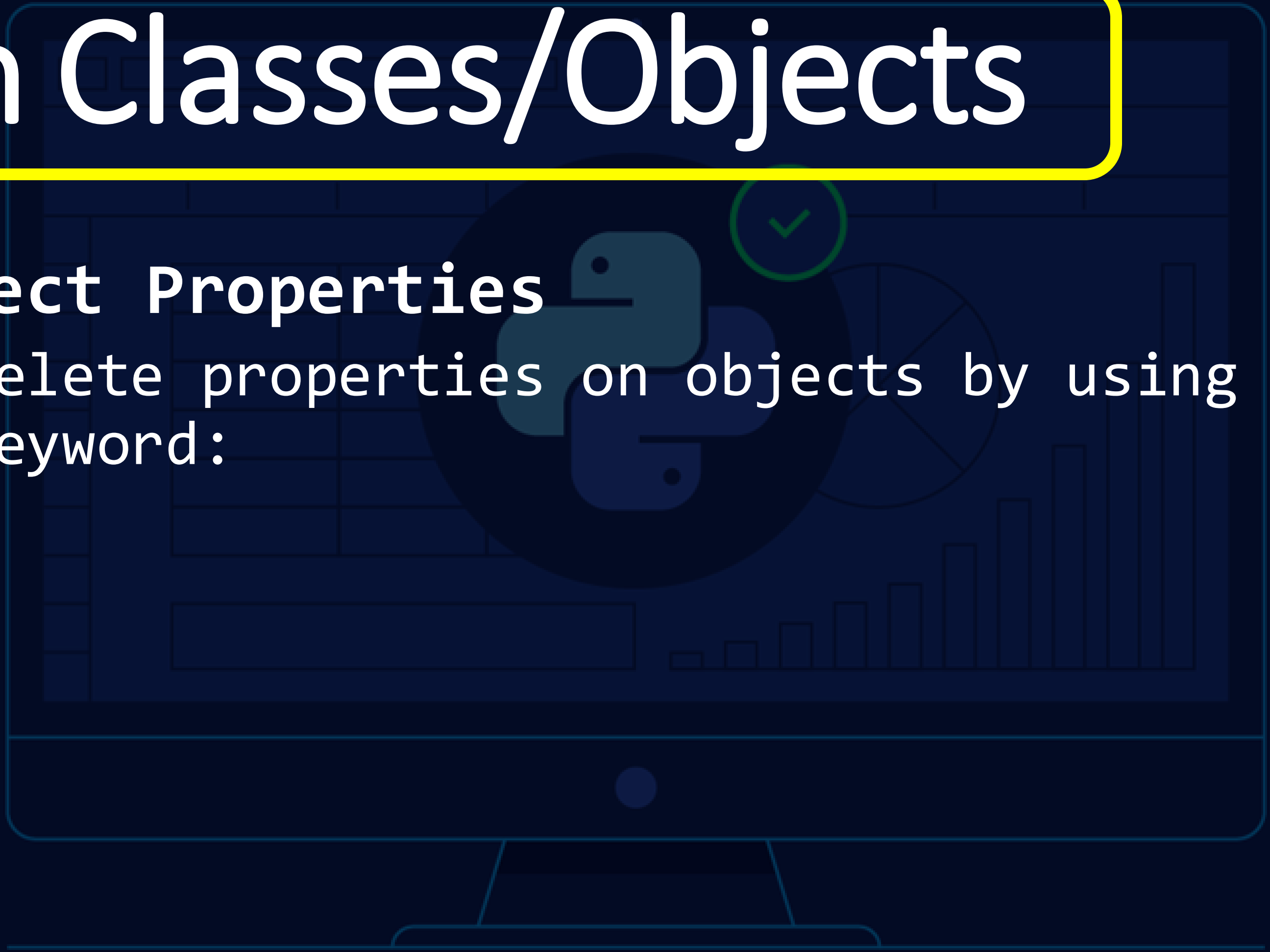


Set the age of p1 to 27:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("Joaquin", 18)

p1.age = 27

print(p1.age)
```

```
27
```

Python Classes/Objects

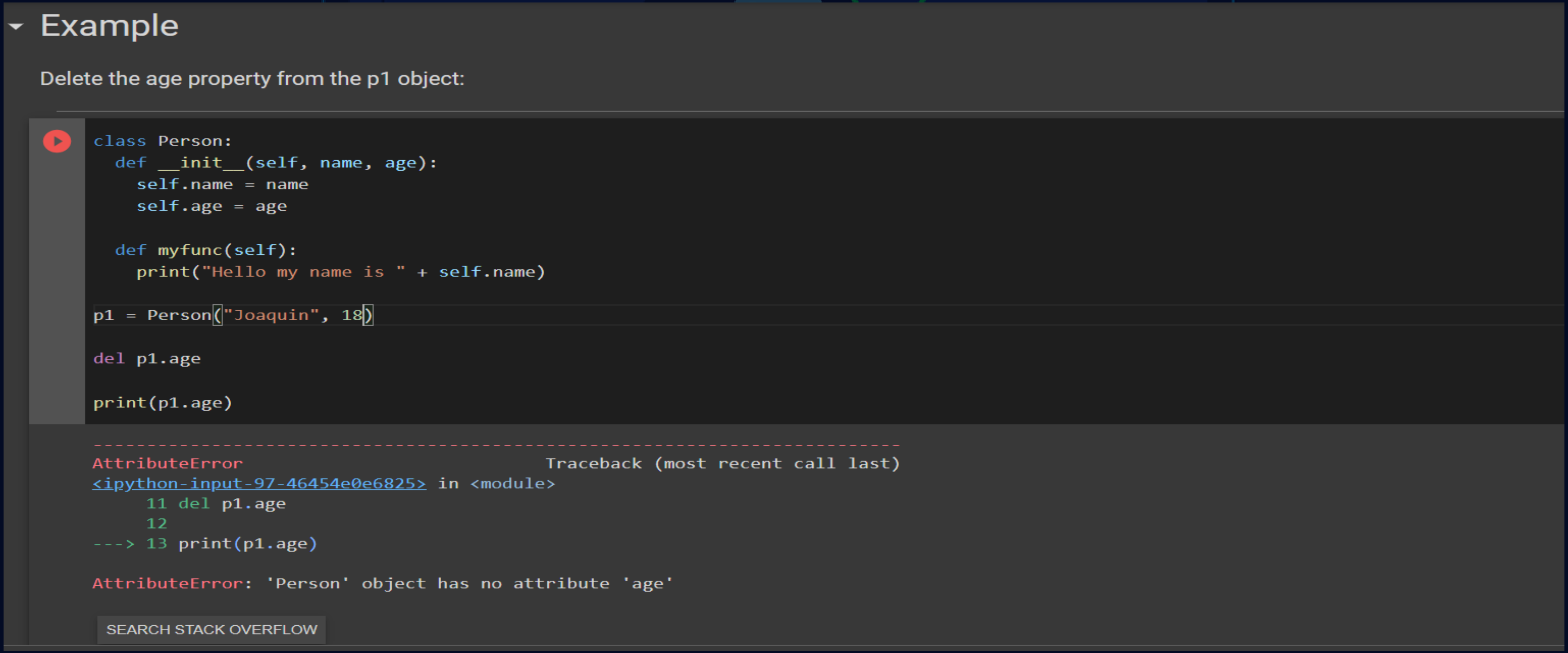


Delete Object Properties

You can delete properties on objects by using the del keyword:

EXAMPLE BELOW

Python Classes/Objects



Example

Delete the age property from the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("Joaquin", 18)

del p1.age

print(p1.age)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-97-46454e0e6825> in <module>
     11 del p1.age
     12
---> 13 print(p1.age)

AttributeError: 'Person' object has no attribute 'age'
```

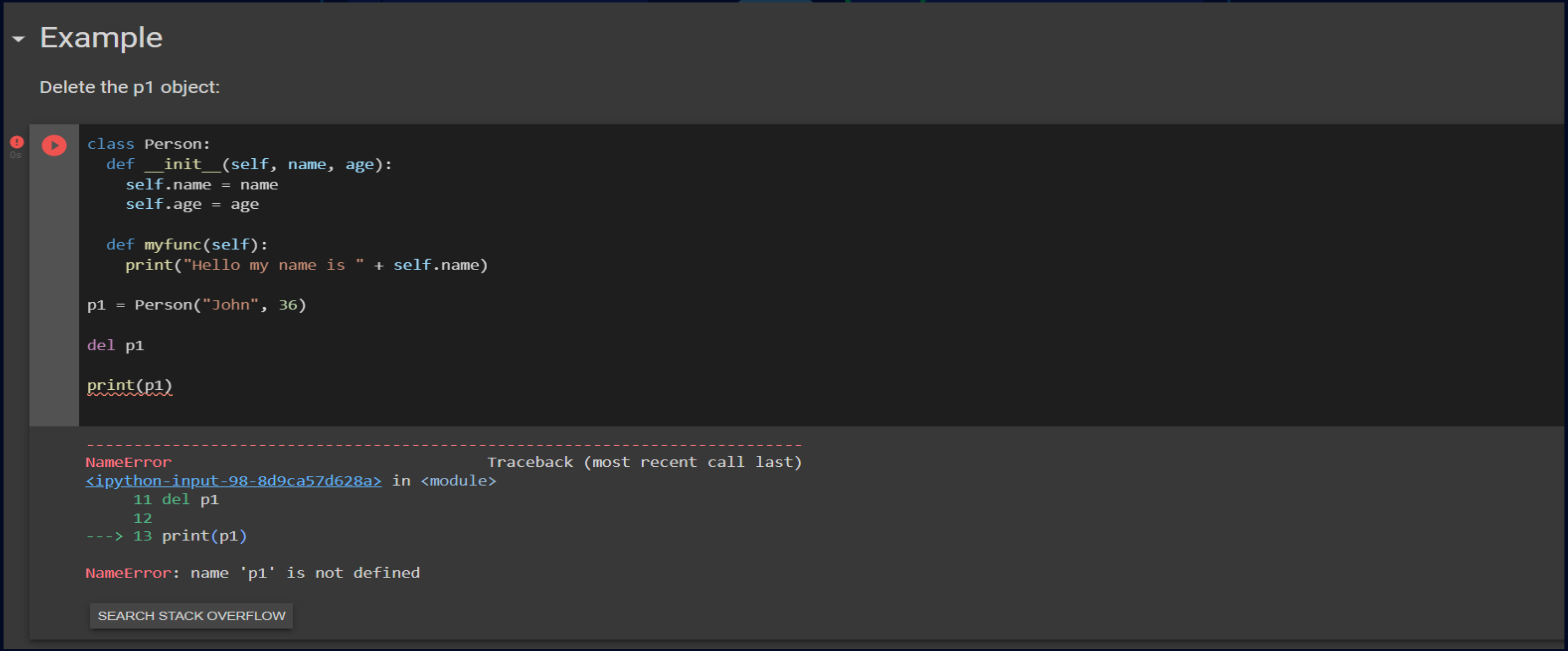
Python Classes/Objects

Delete Objects

You can delete objects by using the del keyword:

EXAMPLE BELOW

Python Classes/Objects



Example

Delete the p1 object:

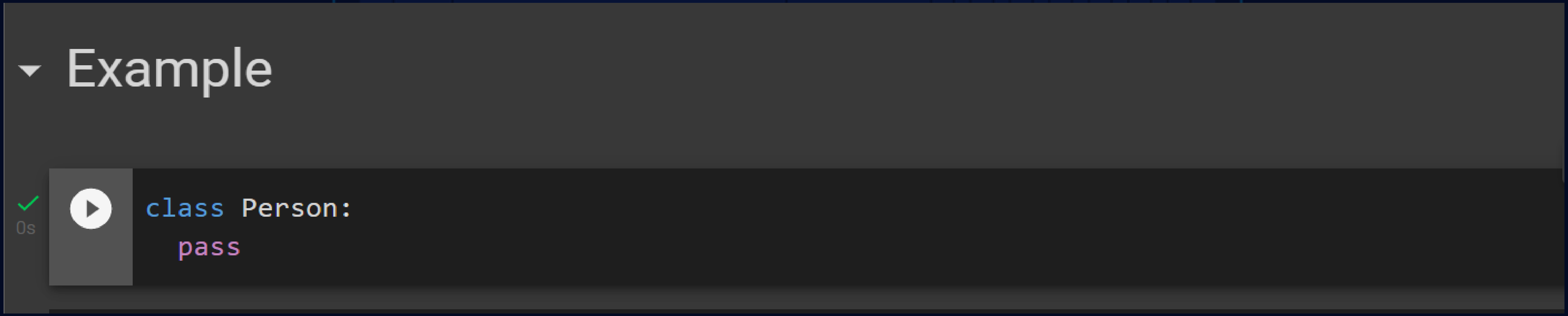
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)

del p1

print(p1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-98-8d9ca57d628a> in <module>
     11 del p1
     12
---> 13 print(p1)

NameError: name 'p1' is not defined
```

Python Classes/Objects

The pass Statement

class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.



Example

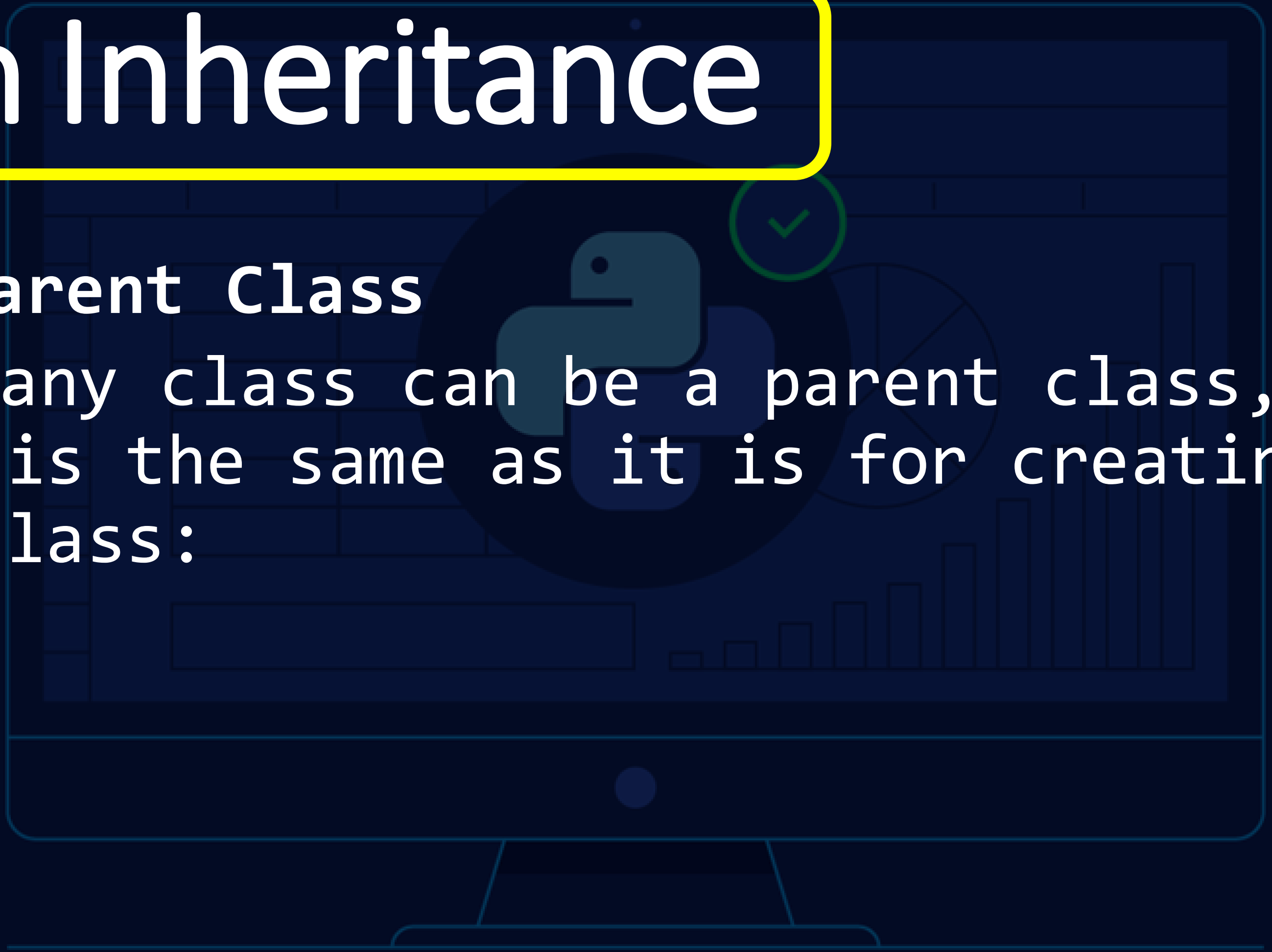
```
class Person:
  pass
```

Python Inheritance

Python Inheritance

- Inheritance allows us to create a class that inherits all of another class's methods and properties.
- The class being inherited from is known as the parent class, sometimes known as the base class.
- A child class is one that inherits from another class, often known as a derived class.

Python Inheritance

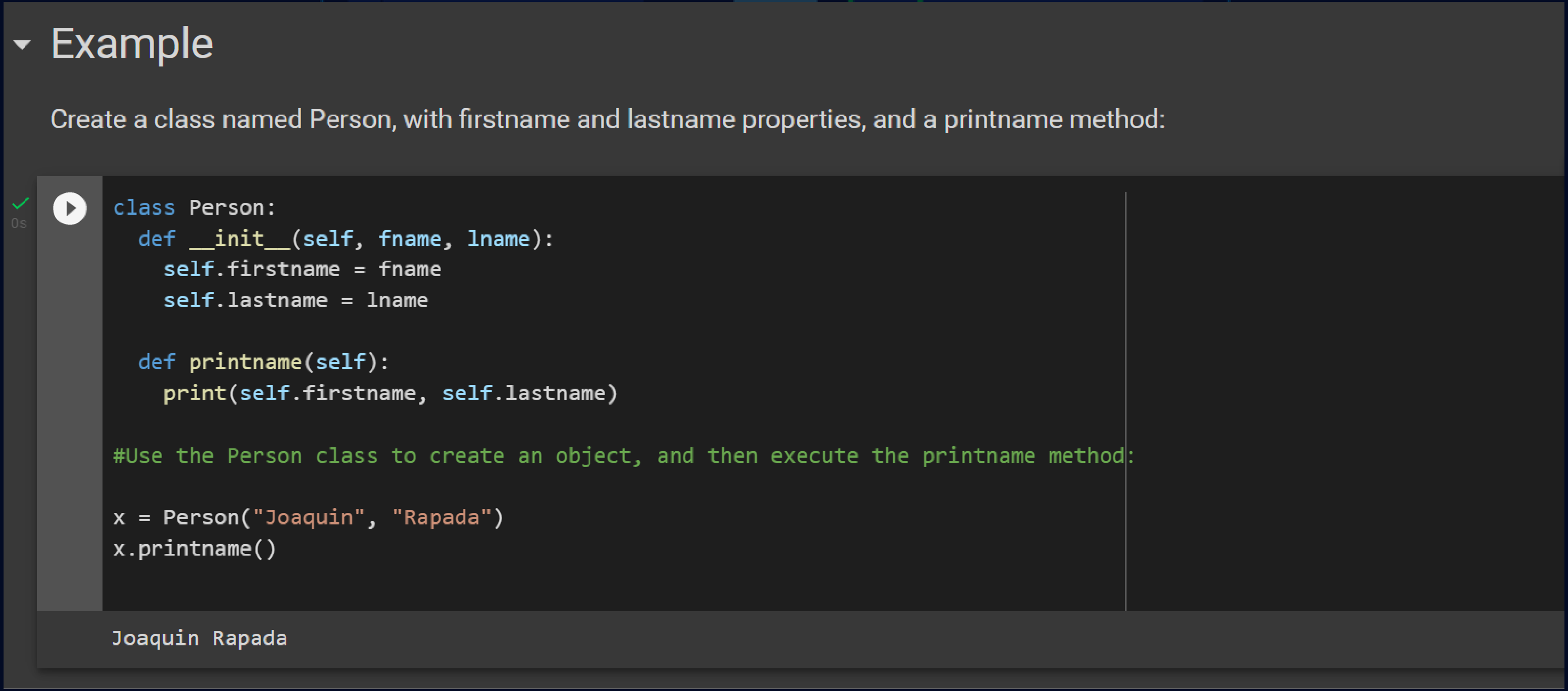


Create a Parent Class

Because any class can be a parent class, the syntax is the same as it is for creating any other class:

EXAMPLE BELOW

Python Inheritance



Example

Create a class named Person, with firstname and lastname properties, and a printname method:

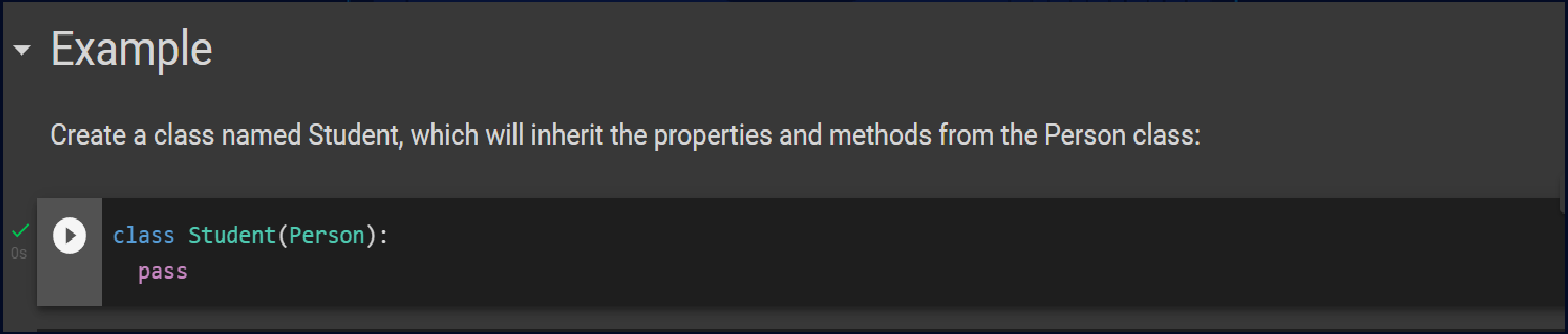
```python
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

#Use the Person class to create an object, and then execute the printname method:

x = Person("Joaquin", "Rapada")
x.printname()
```

```
Joaquin Rapada
```

Python Inheritance

Create a Child Class

Send the parent class as a parameter when constructing the child class to build a class that inherits functionality from another class:



Example

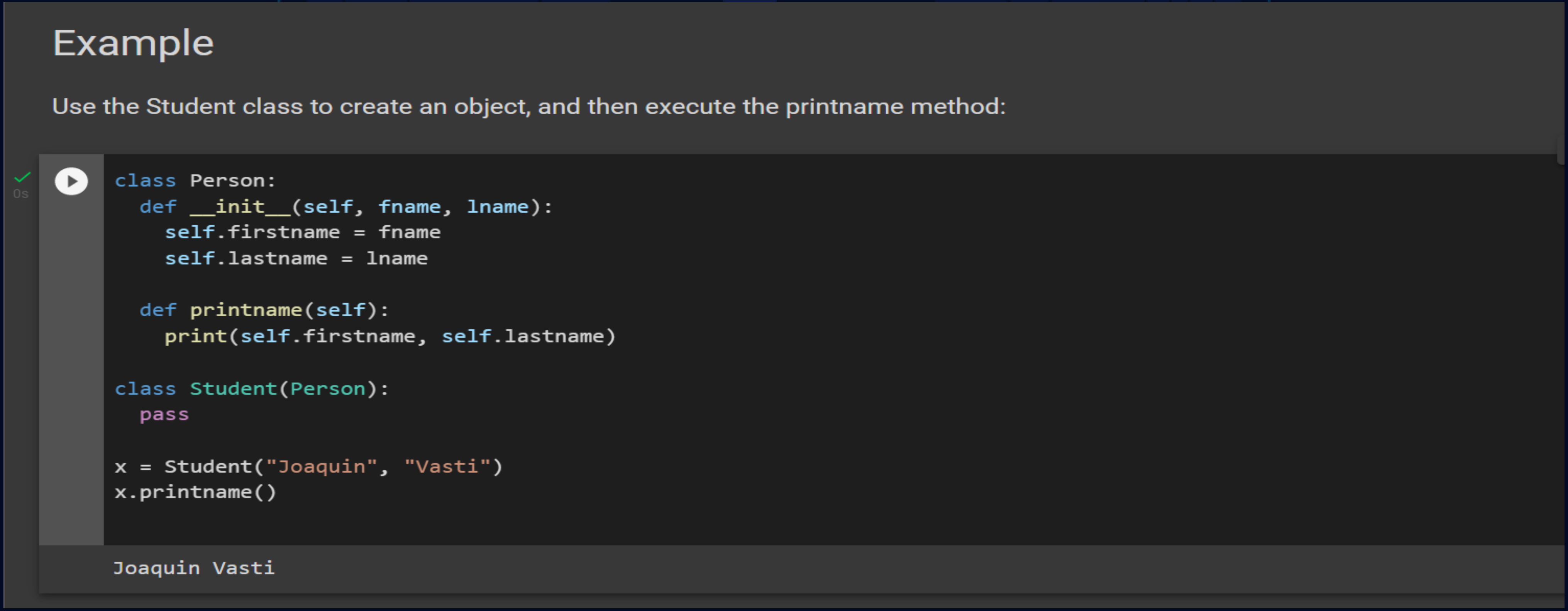
Create a class named Student, which will inherit the properties and methods from the Person class:

```
class Student(Person):
  pass
```

Note: Use the pass keyword when you do not want to add any other properties or methods to the class.

Python Inheritance

The Person class now has the same properties and methods as the Student class.



Example

Use the Student class to create an object, and then execute the printname method:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  pass

x = Student("Joaquin", "Vasti")
x.printname()
```

```
Joaquin Vasti
```

Python Inheritance

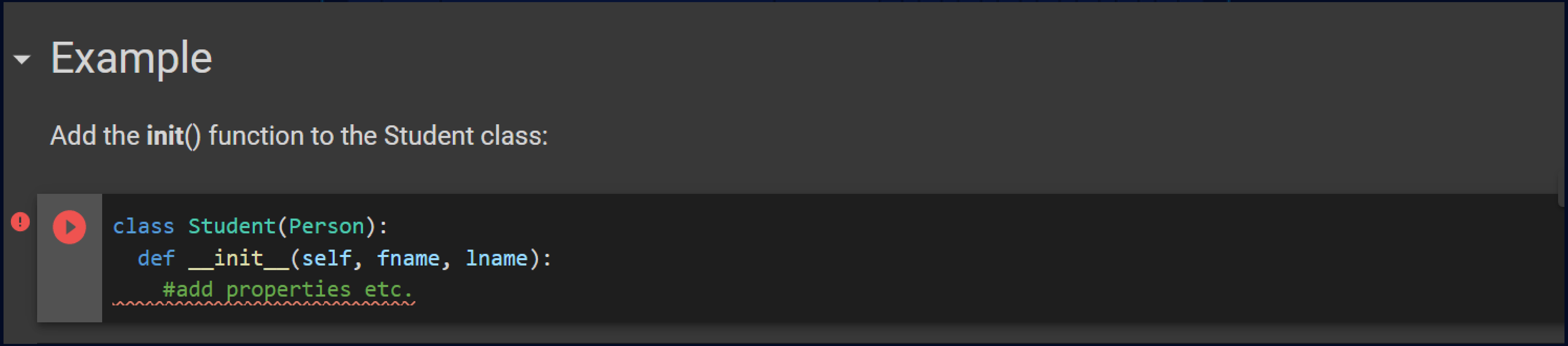
Add the __init__() Function

- So far, we've constructed a child class that inherits its parent's attributes and methods.
- The __init__() function should be added to the child class (instead of the pass keyword).
- The __init__() function is automatically invoked whenever the class is used to create a new object.

Python Inheritance

When you add the `__init__()` function, the child class will no longer inherit the parent's `__init__()` function.

Note: The child's `__init__()` function **overrides** the inheritance of the parent's `__init__()` function.



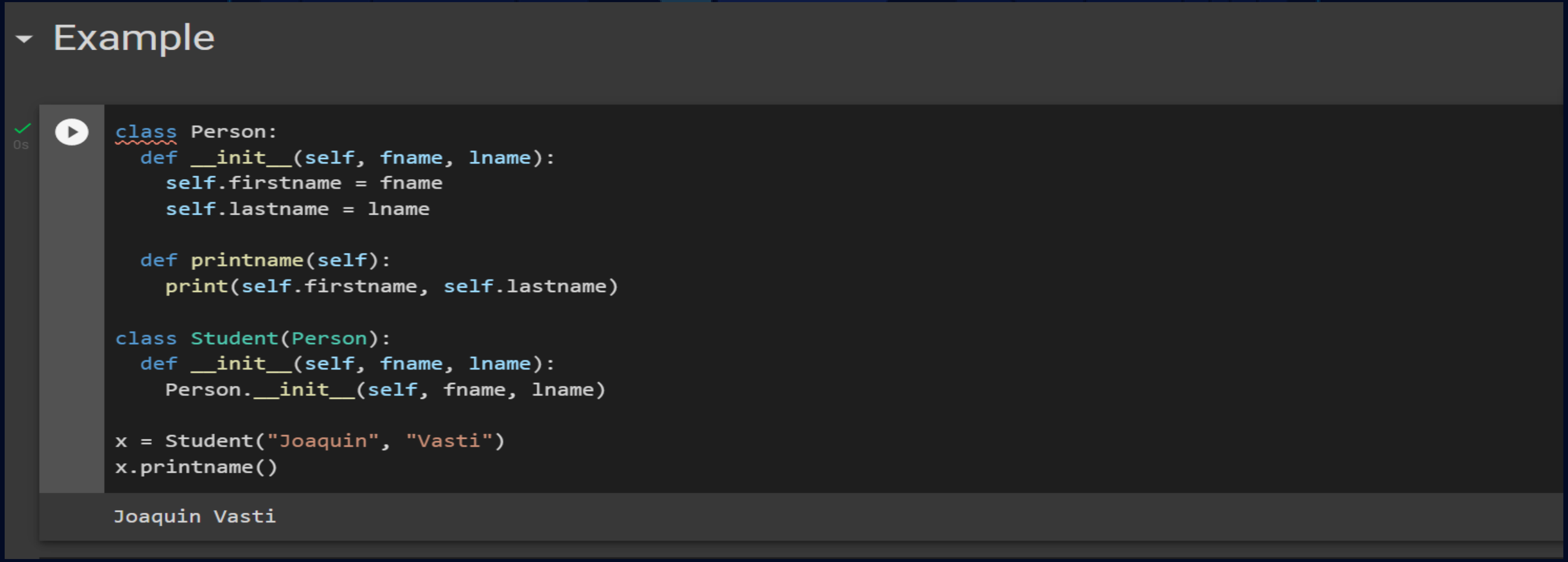
Example

Add the **init**() function to the Student class:

```
class Student(Person):
  def __init__(self, fname, lname):
    #add properties etc.
```

Python Inheritance

To maintain the inheritance of the parent's init () function, add the following call to it:



Example

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    Person.__init__(self, fname, lname)

x = Student("Joaquin", "Vasti")
x.printname()
```

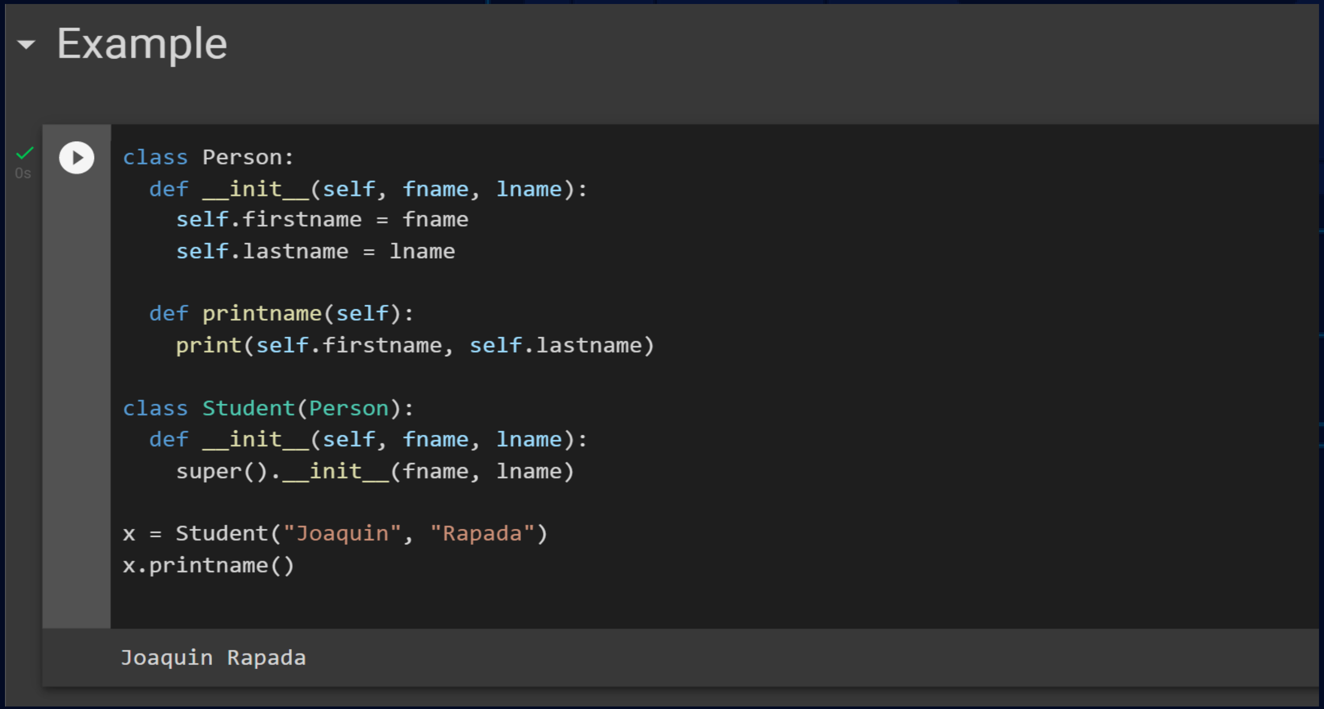
```
Joaquin Vasti
```

Now that we've successfully added the __init__() function and retained the parent class's inheritance, we're ready to add functionality to the __init__() function.

Python Inheritance

Use the super() Function

Python also includes a super() function that will make the child class inherit all of its parent's methods and properties:



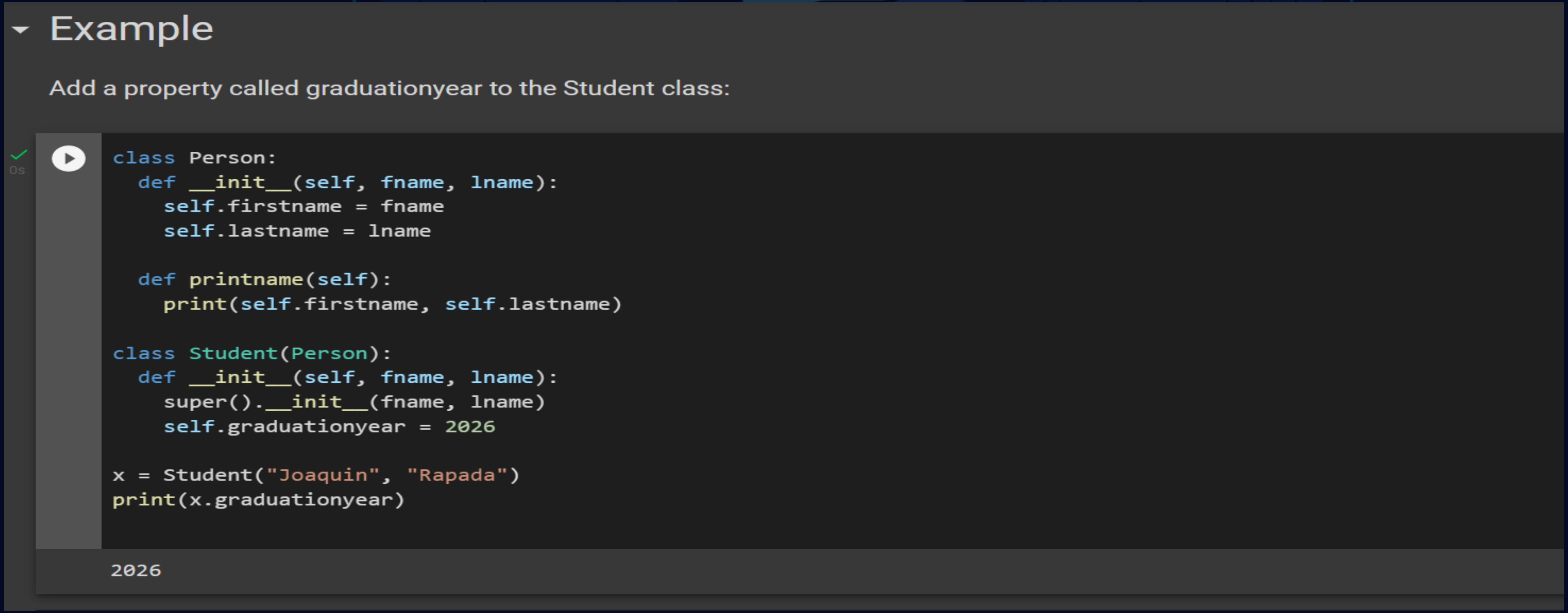
```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)

x = Student("Joaquin", "Rapada")
x.printname()
```

```
Joaquin Rapada
```

You don't have to mention the parent element's name when using the super() function; it will automatically inherit its parent's methods and properties.

Python Inheritance

Add Properties



Example

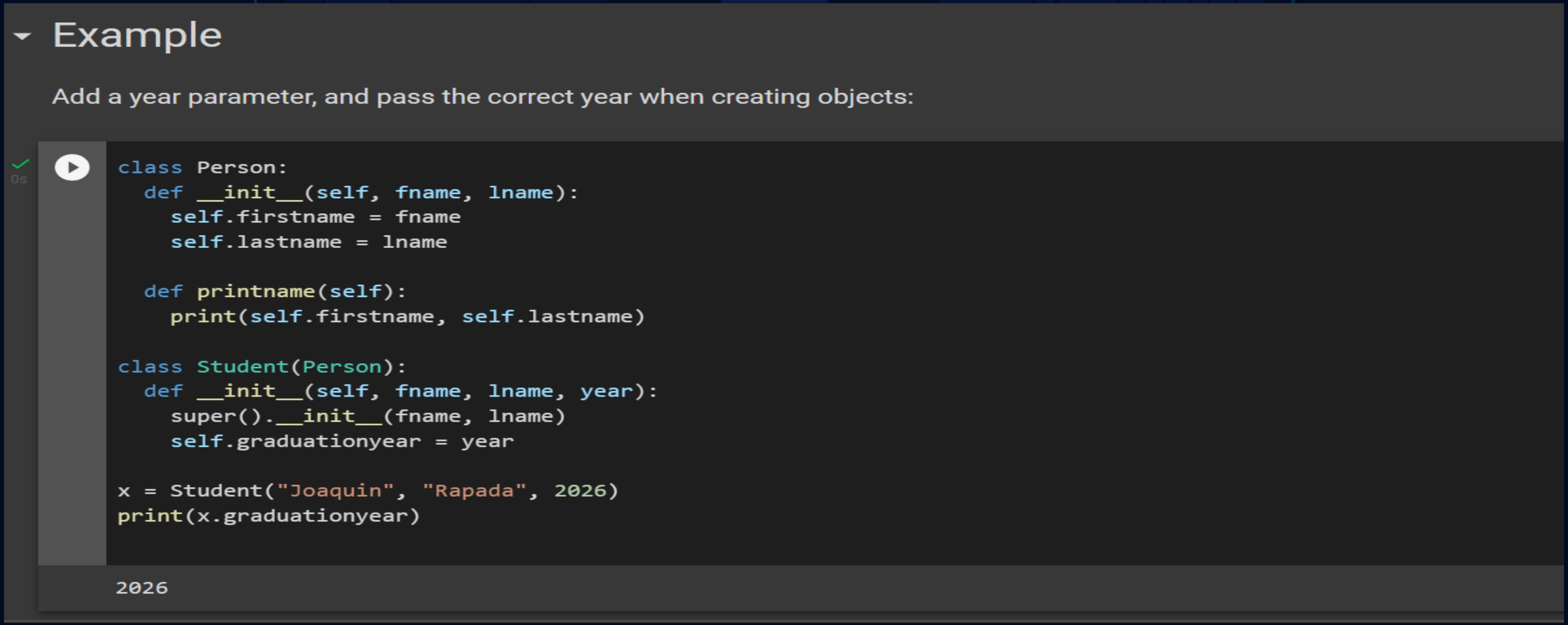
Add a property called graduationyear to the Student class:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname):
    super().__init__(fname, lname)
    self.graduationyear = 2026

x = Student("Joaquin", "Rapada")
print(x.graduationyear)
```

```
2026
```

Python Inheritance

In the example below, the year 2019 should be a variable, and passed into the Student class when creating student objects. To do so, add another parameter in the __init__() function:



Example

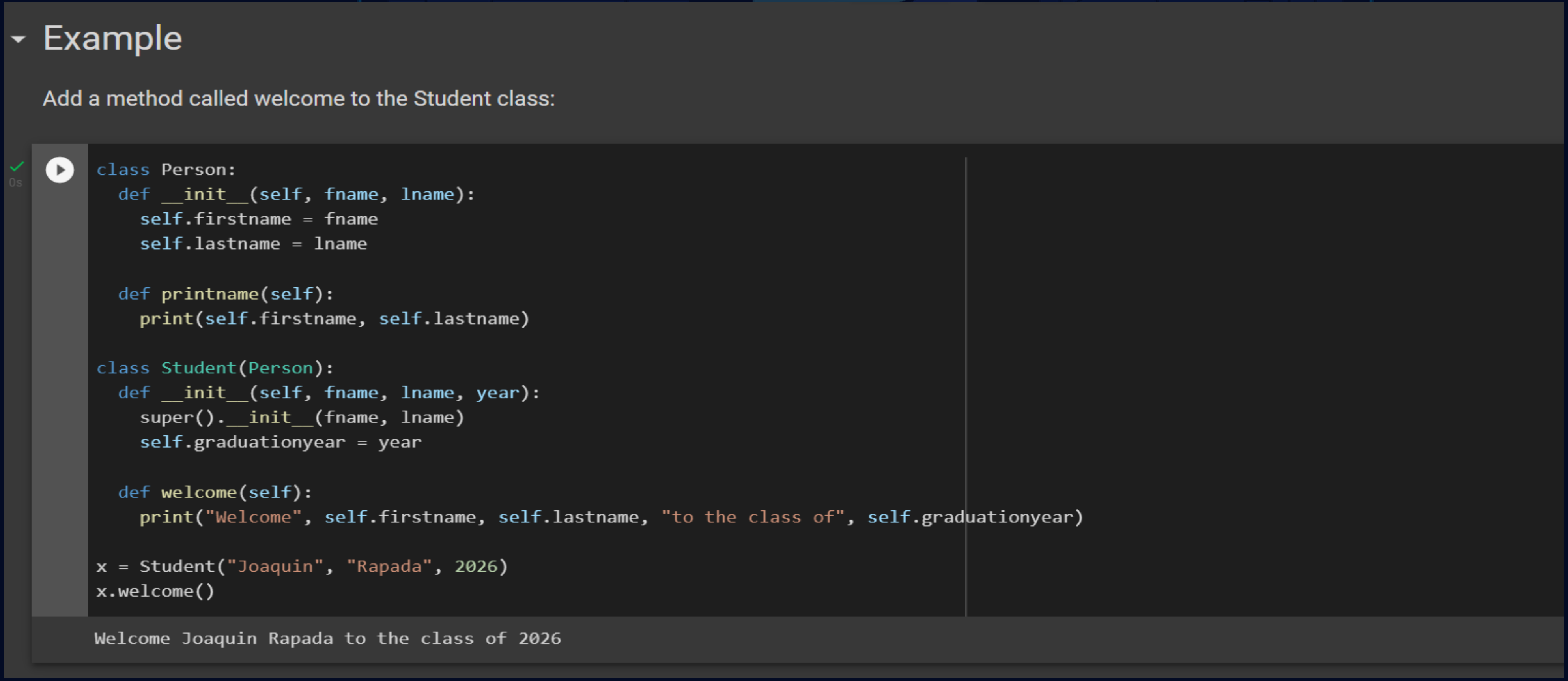
Add a year parameter, and pass the correct year when creating objects:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

x = Student("Joaquin", "Rapada", 2026)
print(x.graduationyear)
```

```
2026
```

Python Inheritance

Add Methods



Example

Add a method called welcome to the Student class:

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

class Student(Person):
  def __init__(self, fname, lname, year):
    super().__init__(fname, lname)
    self.graduationyear = year

  def welcome(self):
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

x = Student("Joaquin", "Rapada", 2026)
x.welcome()
```

```
Welcome Joaquin Rapada to the class of 2026
```

Python Interators

Python Iterators

- An iterator is a collection of values that may be counted.
- An iterator is an object that can be iterated on, which means that you can go over all of the values.
- An iterator is a Python object that implements the iterator protocol, which includes the methods __iter__() and __next__ ().

Python Interators

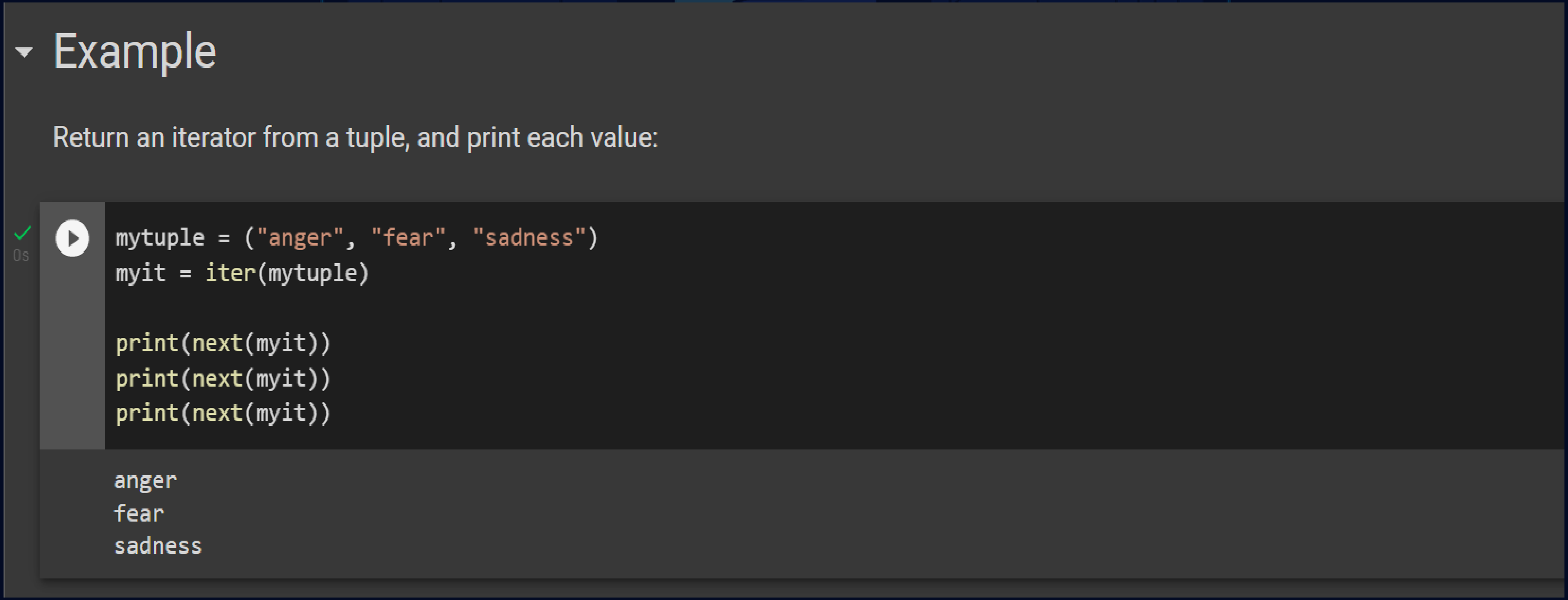
Iterator vs Iterable

- Iterable objects include lists, tuples, dictionaries, and sets. They are iterable containers from which you can obtain an iterator.

- All of these objects have an iter() function that may be used to obtain an iterator:

Python Interators

Iterator vs Iterable



Example

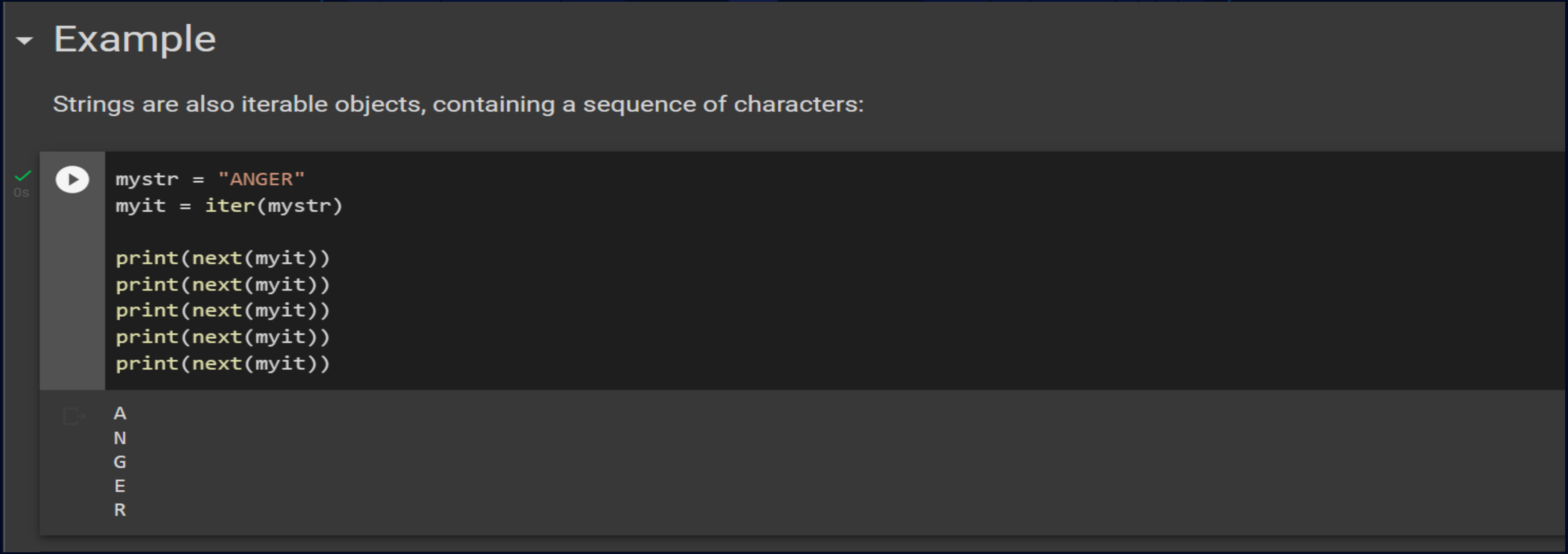
Return an iterator from a tuple, and print each value:

```python
mytuple = ("anger", "fear", "sadness")
myit = iter(mytuple)

print(next(myit))
print(next(myit))
print(next(myit))
```

```
anger
fear
sadness
```

Python Interators

Even strings are iterable objects, and can return an iterator:



Example

Strings are also iterable objects, containing a sequence of characters:

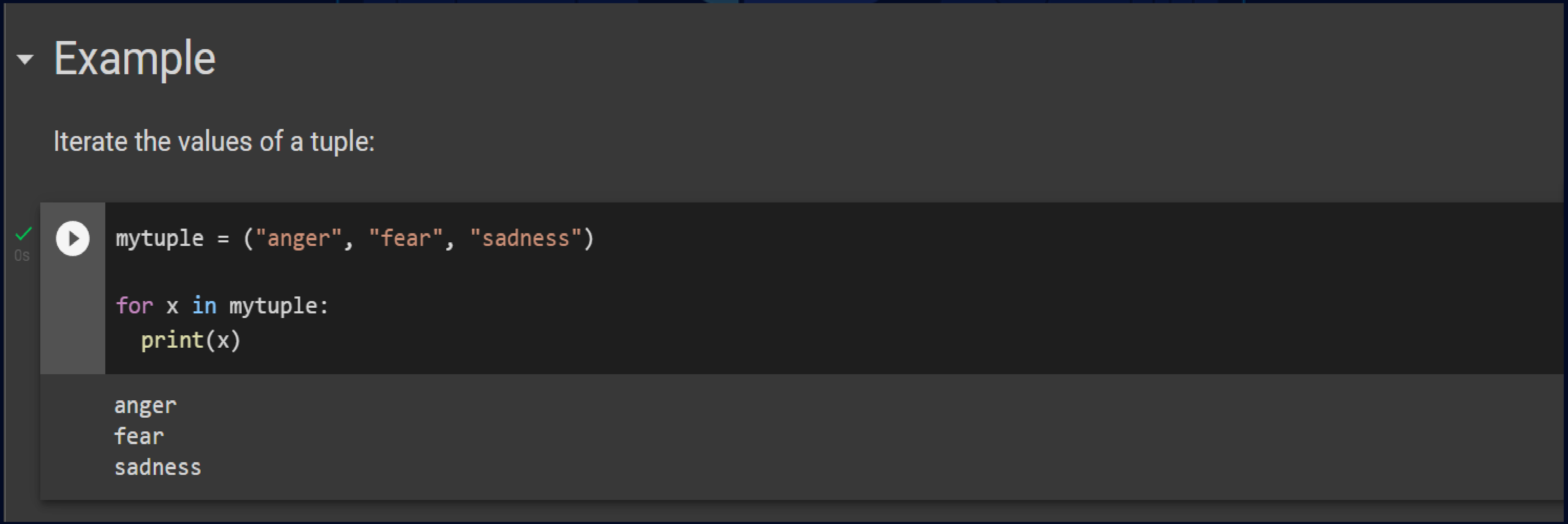
```
mystr = "ANGER"
myit = iter(mystr)

print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
```

```
A
N
G
E
R
```

Python Interators

Looping Through an Iterator

A for loop can also be used to iterate through an iterable object:



Example

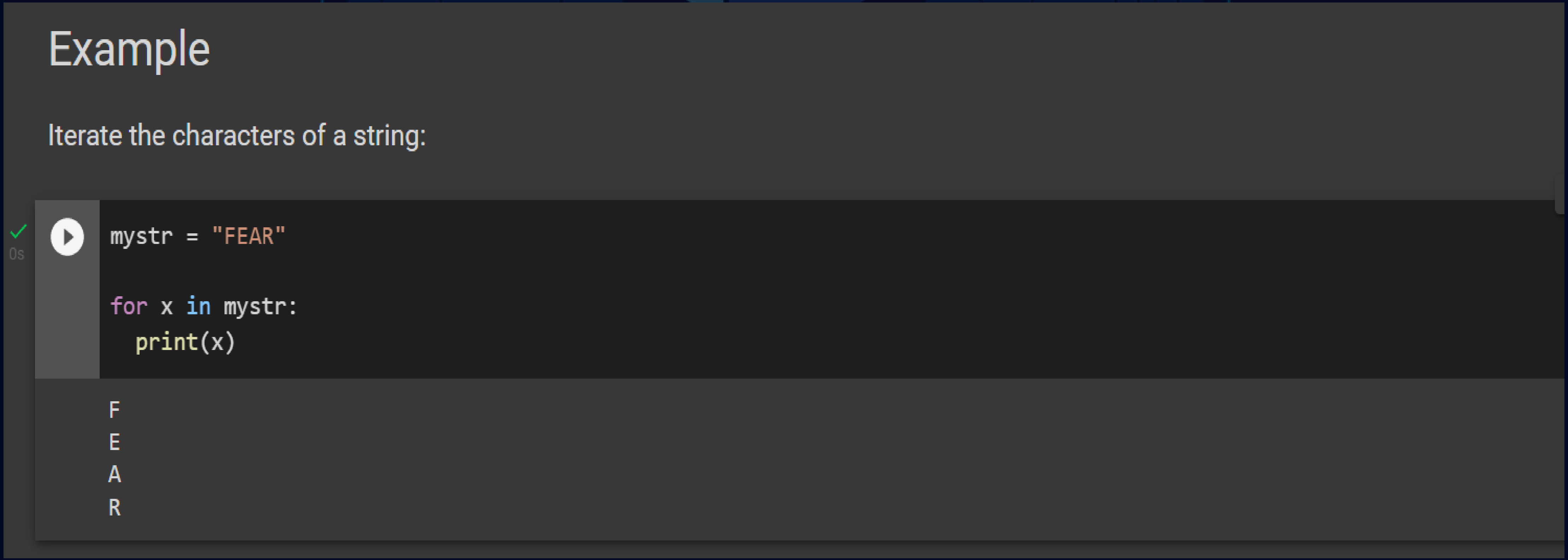
Iterate the values of a tuple:

```
mytuple = ("anger", "fear", "sadness")

for x in mytuple:
  print(x)
```

```
anger
fear
sadness
```

Python Interators

The for loop actually creates an iterator object and executes the next() method for each loop.



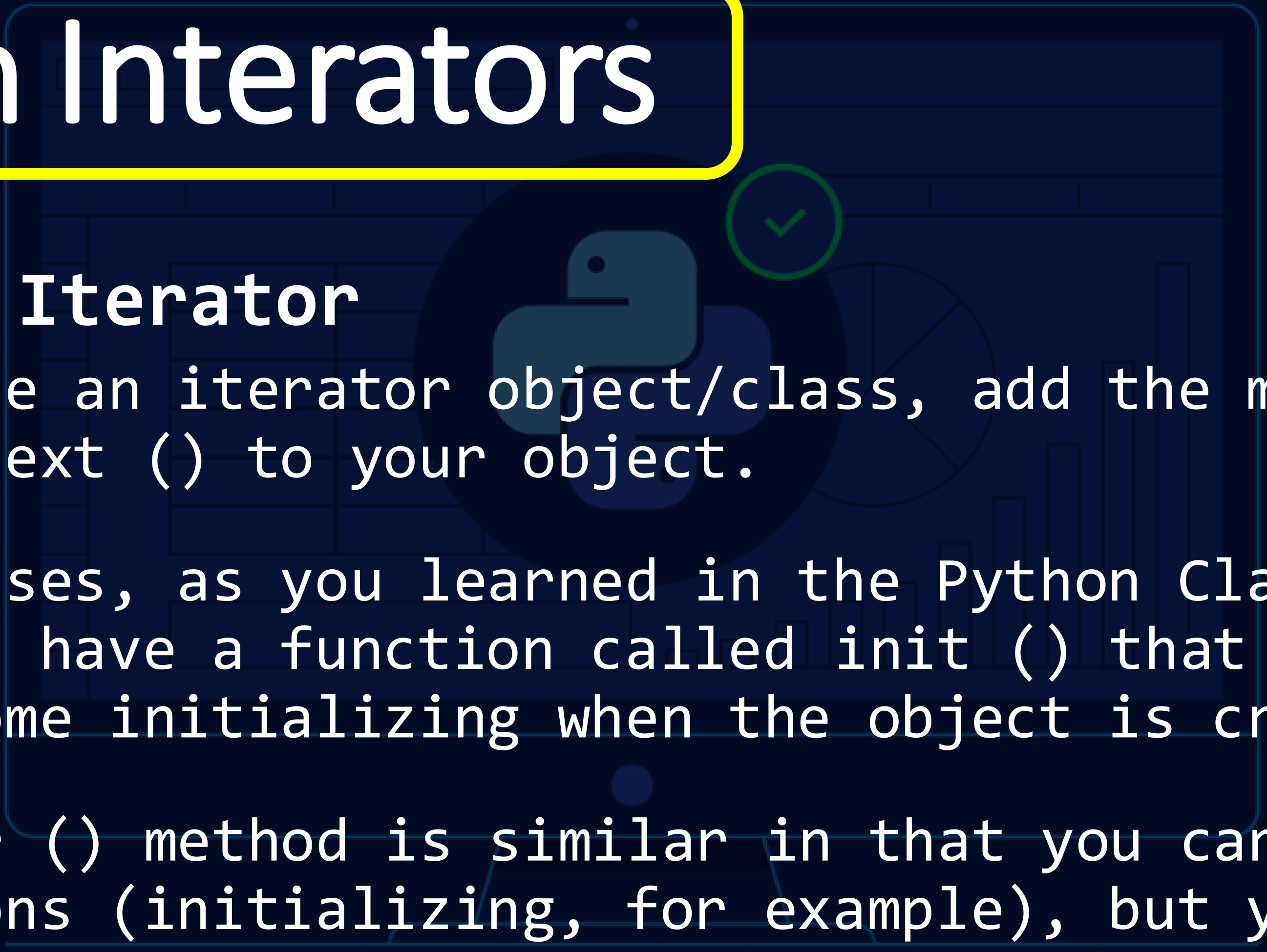
Example

Iterate the characters of a string:

```
mystr = "FEAR"

for x in mystr:
  print(x)
```

```
F
E
A
R
```

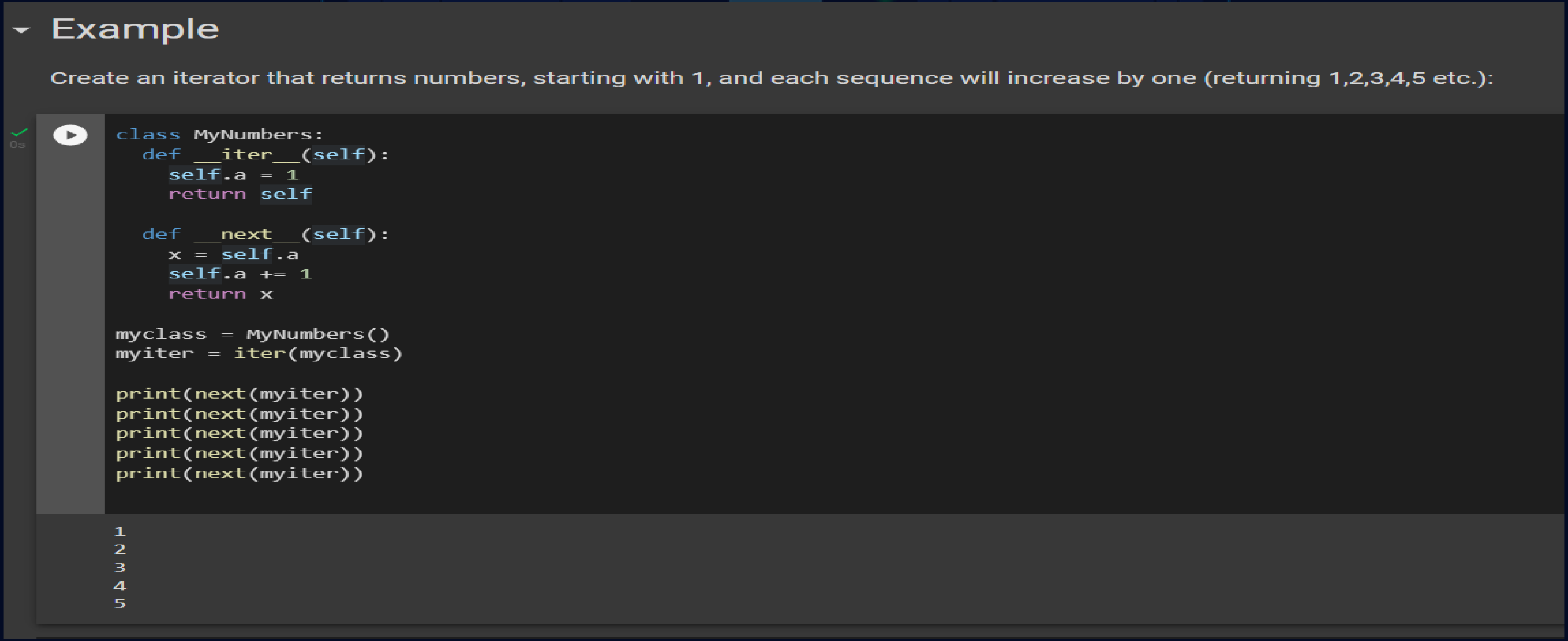
Python Interators



Create an Iterator

- To create an iterator object/class, add the methods iter () and next () to your object.
- All classes, as you learned in the Python Classes/Objects chapter, have a function called init () that allows you to do some initializing when the object is created.
- The iter () method is similar in that you can perform operations (initializing, for example), but you must always return the iterator object itself.
- You can also perform operations with the next () method, which must return the next item in the sequence.

Python Interators



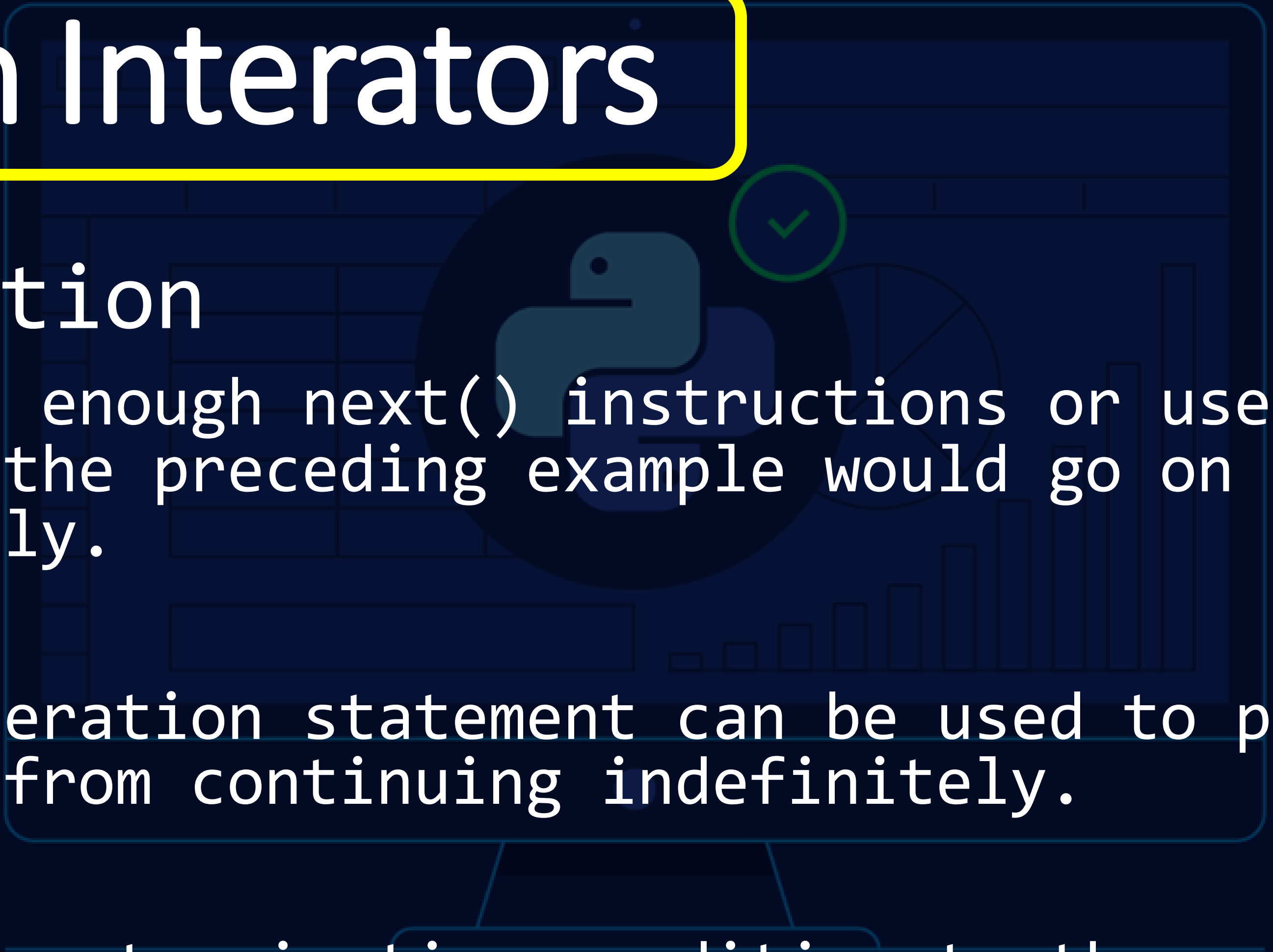
Example

Create an iterator that returns numbers, starting with 1, and each sequence will increase by one (returning 1,2,3,4,5 etc.):

```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    x = self.a
    self.a += 1
    return x

myclass = MyNumbers()
myiter = iter(myclass)

print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
print(next(myiter))
```

```
1
2
3
4
5
```

Python Interators



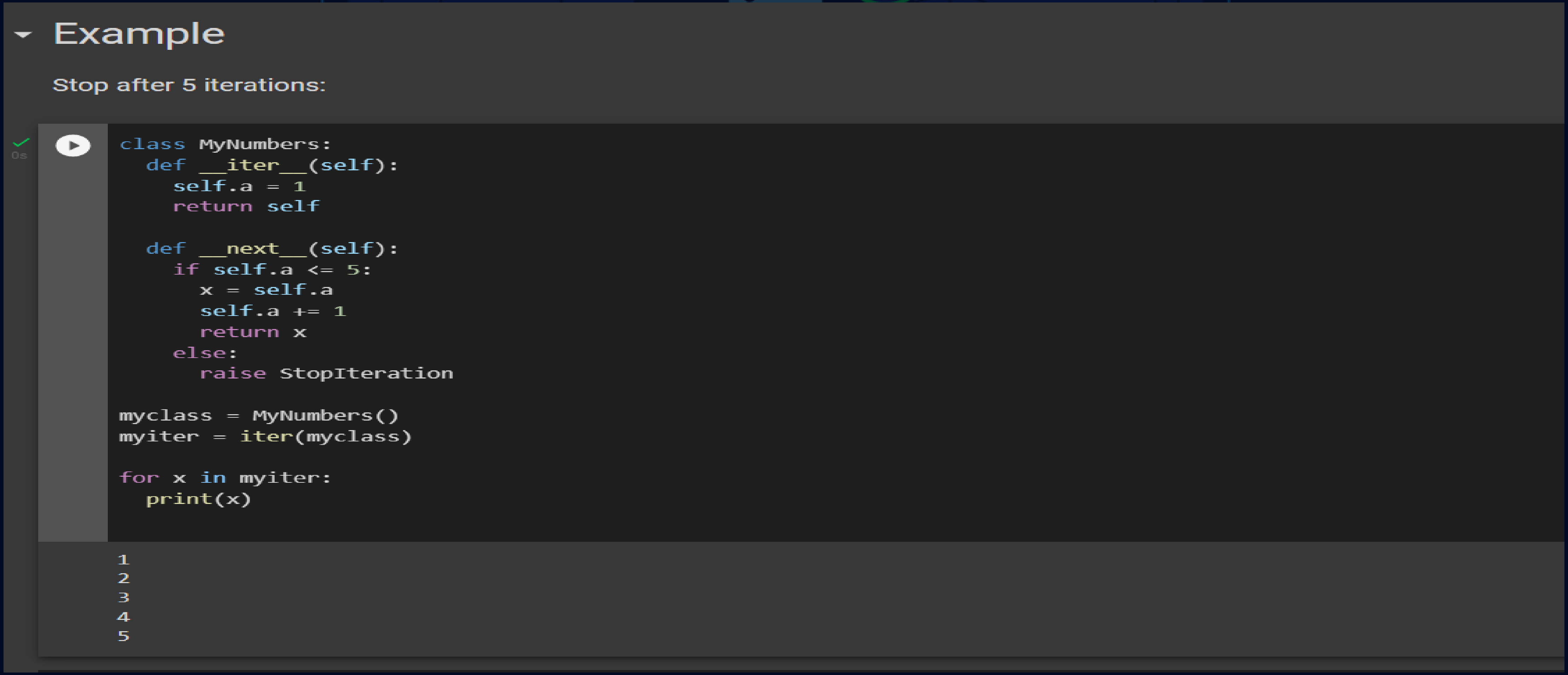
StopIteration

- If you had enough next() instructions or used it in a for loop, the preceding example would go on indefinitely.

- The StopIteration statement can be used to prevent the iteration from continuing indefinitely.

- We may add a terminating condition to the __next__() method to raise an error if the iteration is repeated a certain amount of times:

Python Interators



Example

Stop after 5 iterations:

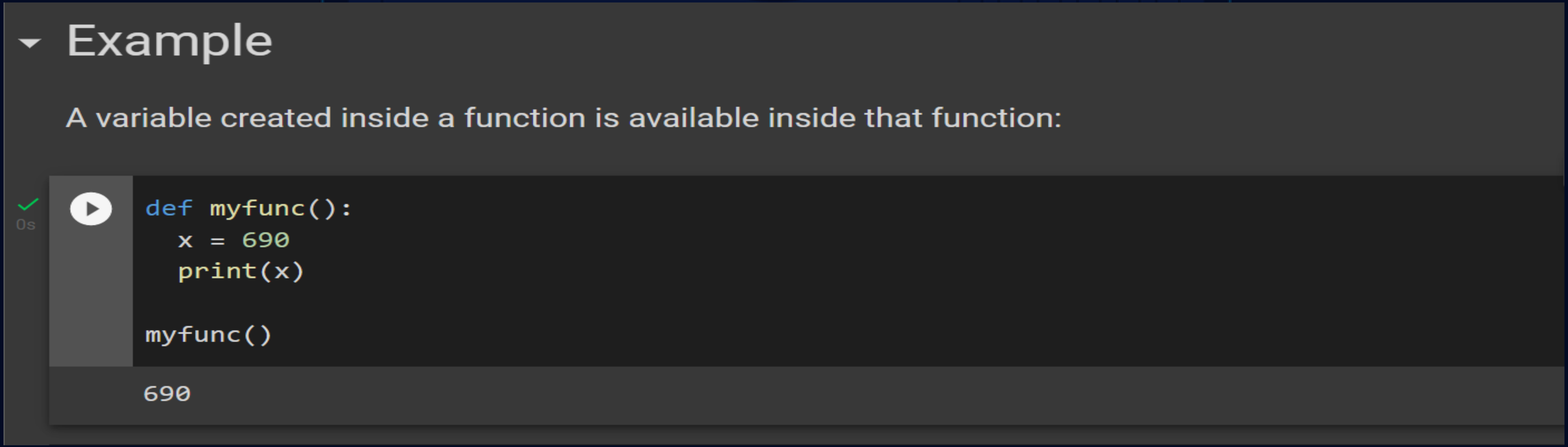
```
class MyNumbers:
  def __iter__(self):
    self.a = 1
    return self

  def __next__(self):
    if self.a <= 5:
      x = self.a
      self.a += 1
      return x
    else:
      raise StopIteration

myclass = MyNumbers()
myiter = iter(myclass)

for x in myiter:
  print(x)
```

```
1
2
3
4
5
```

Python Scope

Local Scope

A variable generated within a function is part of the function's local scope and can only be utilized within that function.



Example

A variable created inside a function is available inside that function:

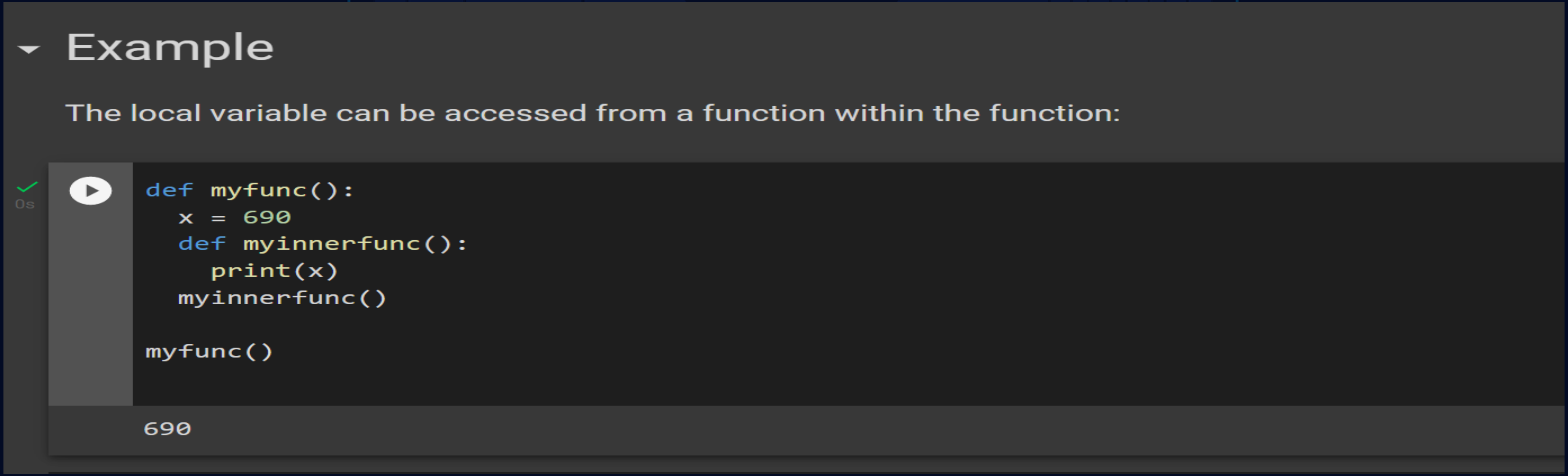
```
def myfunc():
  x = 690
  print(x)

myfunc()
```

```
690
```

Python Scope

Function Inside Function

As shown in the preceding example, the variable x is not available outside of the function, but it is available to any function within the function:



Example

The local variable can be accessed from a function within the function:

```
def myfunc():
  x = 690
  def myinnerfunc():
    print(x)
  myinnerfunc()

myfunc()
```

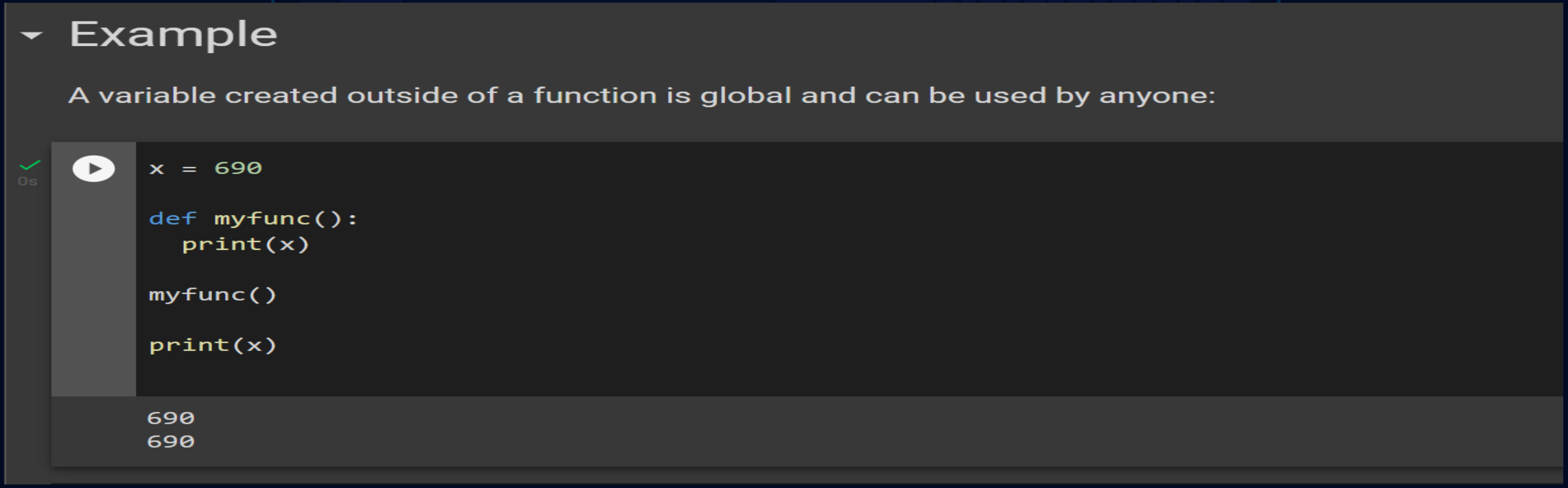
```
690
```

Python Scope

Global Scope

A variable that is created in the main body of the Python code is a global variable that is part of the global scope.

Global variables are accessible from any scope, both global and local.



Example

A variable created outside of a function is global and can be used by anyone:

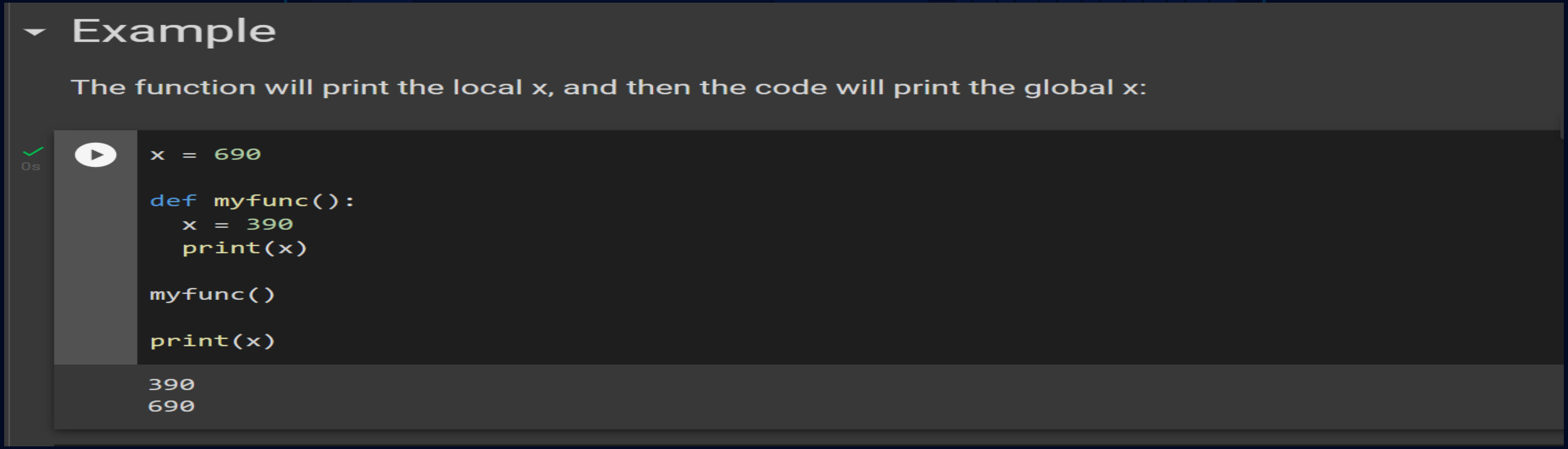
```
x = 690

def myfunc():
  print(x)

myfunc()

print(x)
```

```
690
690
```

Python Scope

Naming Variables

If you use the same variable name inside and outside of a function, Python will interpret them as two distinct variables, one in the global scope (outside the function) and one in the local scope (inside the function):



Example

The function will print the local x, and then the code will print the global x:

```
x = 690

def myfunc():
  x = 390
  print(x)

myfunc()

print(x)
```

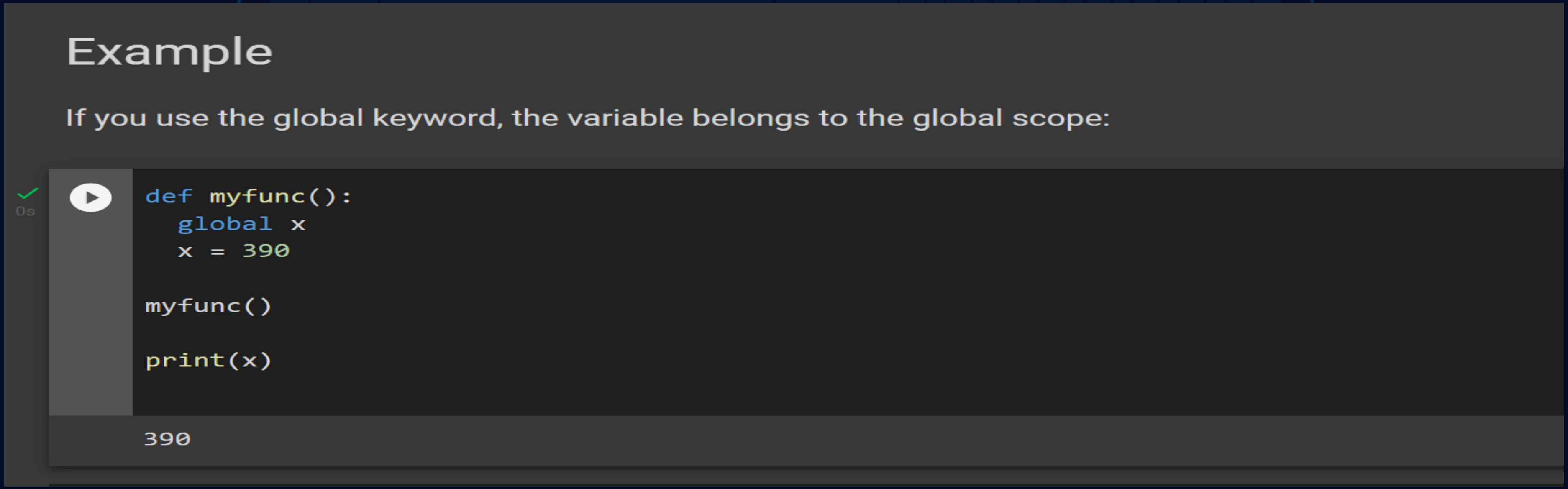
```
390
690
```

Python Scope

Global Keyword

You can use the global keyword if you need to construct a global variable but are trapped in the local scope.

The variable is made global by using the global keyword.



Example

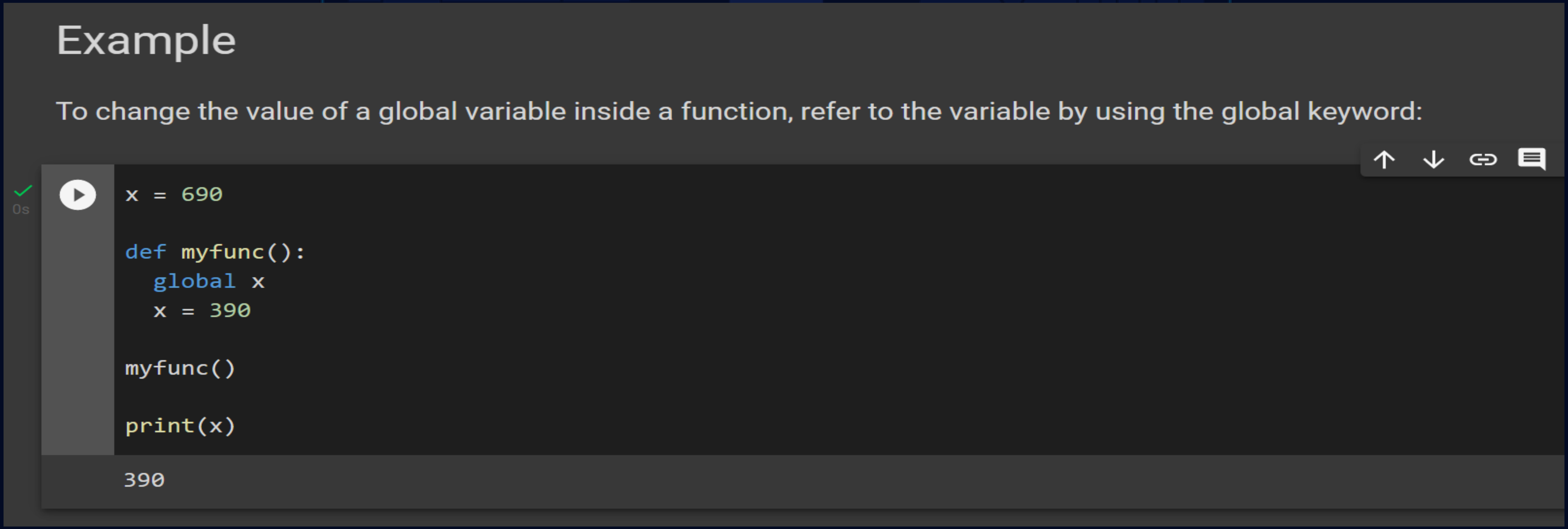
If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = 390

myfunc()

print(x)
```

```
390
```

Python Scope

If you want to change a global variable within a function, use the global keyword.



Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```python
x = 690

def myfunc():
  global x
  x = 390

myfunc()

print(x)
```

```
390
```

END OF VERSION TWO

THANK YOU!

CHINESE NAME: 喬萬斯

STUDENT ID: 4110E234

English Name: Joaquin Vasti R. Rapada

nickname: Wax

email: vastiplayer@gmail.com

GITHUB: https://github.com/4110E234/cs20220921/tree/main/python