Module 2 - Functions

**1. Simple Functions**

**1.1 Intro Python**

Jupyter Notebook: MOD02\_1-3.1\_Intro\_Python.ipynb

Functions Arguments & Parameters

* **Creating a simple Function with a parameter**
* Exploring Functions with return values
* Creating Functions with multiple parameters
* Sequence in python

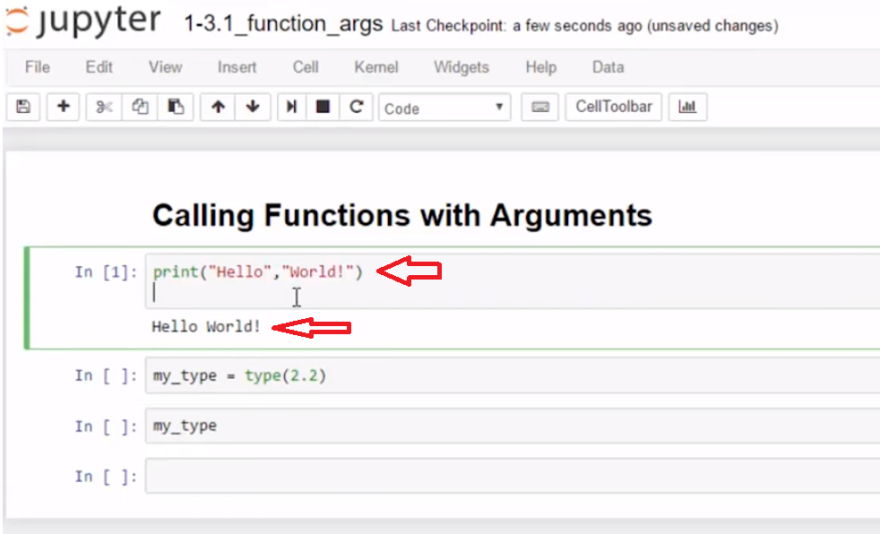
Student will be able to

* **create functions with a parameter**

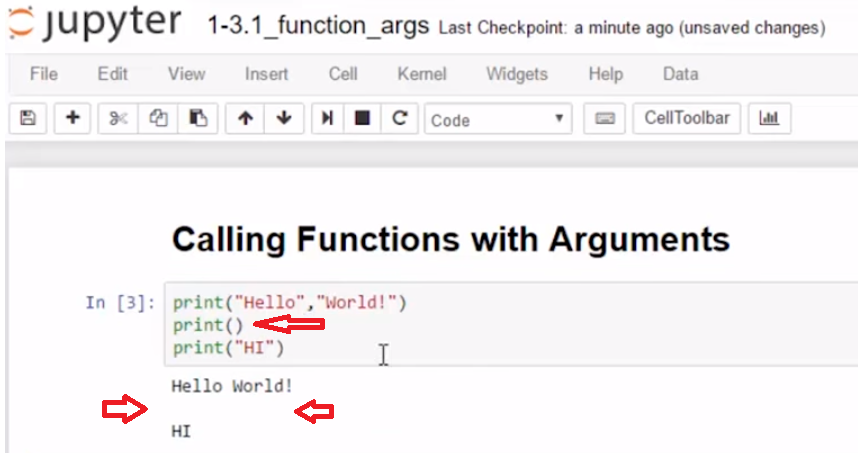
**1.2 Call Function with arguments**

**Video:** **CallFunctionWithArgumentsV1.mp4**

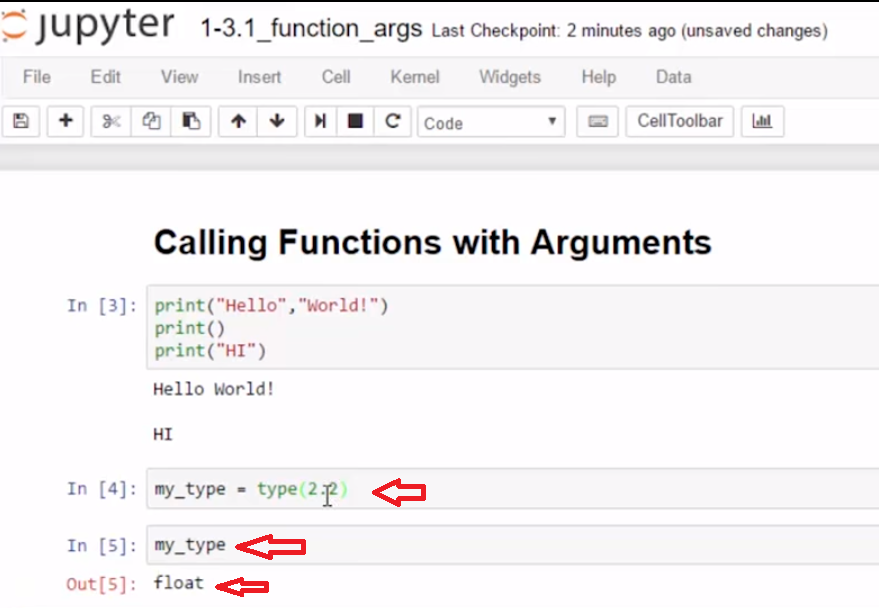
Let's look at some examples of functions that will take arguments. **The print statement can take zero to many arguments.** And here's an example of the print statement with two arguments. **The first argument is the string Hello and then there is a comma which separates the next argument which is World string.** And so, print statement that just sends whatever strings are in there to standard out so, we can go and run that cell. And we see, Hello World like we expect



**but print is unique in that, It can take a variable amount of arguments including zero arguments. So, let's run that cell now and we see now there's a extra space there.** And we can even call print with one argument. And then, it's a little easier to see, right there that we have a blank line that was printed when we sent it zero argument. **Print can take two arguments or no arguments or one arguments and can take up to, almost infinite amount of arguments.**



**In this second example, we have the type function and type is not so forgiving. It will take exactly one argument of an object and tell you it's type.** In this case, we put a number 2.2 in there and another thing about ***type, it doesn't have the same behavior as print. It doesn't put its answer out to the standard output it just returns a value.*** And so, right here I'm storing that value in a variable called my\_type. And so, let's look at how type will work. So if I hit Enter then, **type will return it's value into the my\_type variable and then we can run my\_type. You see what was in there, yep, 2.2 that's a float.**



So, we're seeing that arguments can be sent to functions in many different ways where there's a required amount, one or as any other specific number of arguments. Or they can be variable like we have here with the print statement.

Python allows us to create **User Defined Functions** and provides many **Built-in Functions** such as **print()**

* **print()** can be called using arguments (or without) and sends text to standard output, such as the console.
* **print()** uses **Parameters** to define the variable Arguments that can be passed to the Function.
* **print()** defines multiple string/numbers parameters which means we can send a long list of Arguments to **print()**, separated by commas.
* **print()** can also be called directly with just its name and empty parenthesis and it will return a blank line to standard output

# Example

print('Hello World!', 'I am sending string arguments to print ')

student\_age = 17

student\_name = "Hiroto Yamaguchi"

print(student\_name,'will be in the class for',student\_age, 'year old students.')

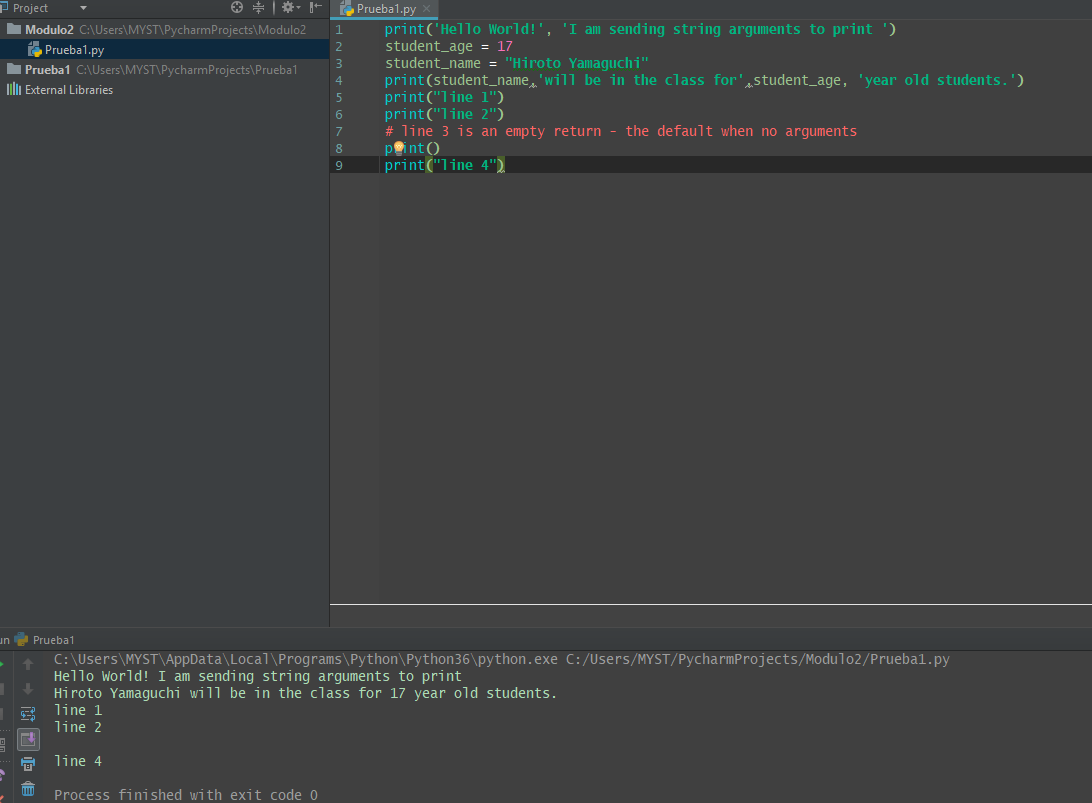
print("line 1")

print("line 2")

# line 3 is an empty return - the default when no arguments

print()

print("line 4")



# Task 1

## Passing Arguments to print()

### Many Arguments can be passed to print

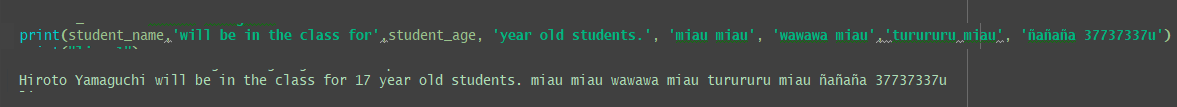
* update the print statement to use **print()** with **8** or more arguments

#[ ] increase the number of arguments used in print() to 8 or more

student\_age = 17

student\_name = "Hiroto Yamaguchi"

print(student\_name,'will be in the class for',student\_age, 'year old students.')



**1.3 Create Simple Function**

**Video:** **CreateSimpleFunctionV2.mp4**

# Concept: Simple Functions

## Create a simple Function

Creating user defined functions is at the core of computer programming. Functions enable code reuse and make code easier to develop and maintain.

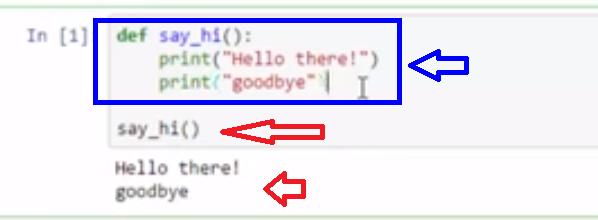
**A simple function can be created with the def keyword, function name, followed by parentheses and a semicolon.** **So here we have two simple functions.** And this first one, say hi, we're going to use the keyword **def which tells Python we are defining a function and then we have the function name**. It just has to start with a letter. We use the standard of lower case and underscore separating words, that is not required, just our standard, and then parenthesis and a colon, and that is your function definition right there.



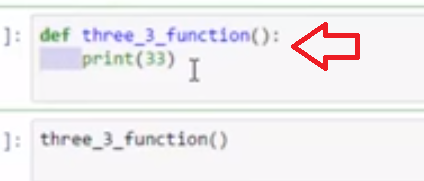
And then you put code underneath, and we'll put many different types of pieces of code in our functions, but right now we're just gonna stick with some simple print statements. **The indentation is four spaces. I can then end the function by ending the indentation.** So anywhere where either the cell just ends, or if I put something else in the cell.



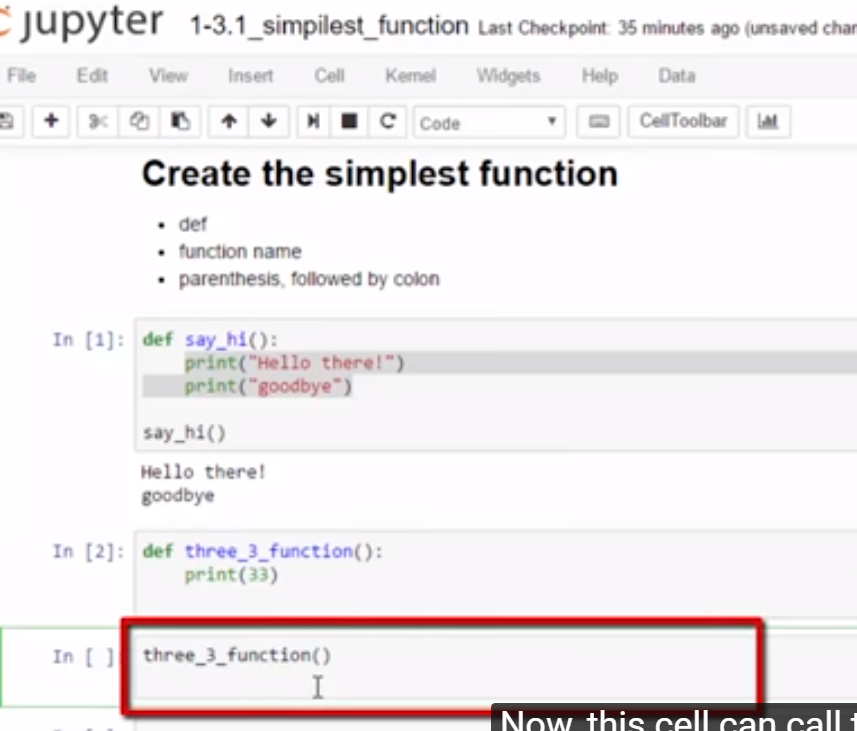
And in this case, I'm going to call the function**. And because this indentation has ended, the function only goes to this point here.** **So to call the function, I use the function name and parenthesis, and then I'm just gonna use this keyboard shortcut Shift+Enter. And it runs the function which prints these two lines, Hello there, goodbye.**



And then I can look at another example where I can call a function from another cell. And in this one, **three\_3\_function, it just prints the number 33. I use a def statement to tell Python it's a function.** I put a function name, and then I put my parenthesis and colon. And I add my function definition again, **four spaces** and then function code. I need to run the function before I can call it though. Why do I run it? Because that puts it in memory.



Now, **this cell can call the function because it's in memory**, and I just use the function name followed by parenthesis, and it prints 33. So those are simple functions, we'll write many more functions in this course.



### basics of a user defined function

* define a function with **def**
* use indentation (4 spaces)
* define parameters
* optional parameters
* **return** values (or none)
* function scope (basics defaults)

### def some\_function():

use the **def** statement when creating a **function**

* use a function name that **starts with a letter** or underscore (usually a lower-case letter)
* function names can contain **letters, numbers or underscores**
* parenthesis **()** follow the function name
* a colon **:** follows the parenthesis
* the code for the function is indented under the function definition (use 4 spaces for this course)

def some\_function():

#code the function tasks indented here

The **end of the function** is denoted by returning to **no indentation**

# Example

# defines a function named say\_hi

def say\_hi():

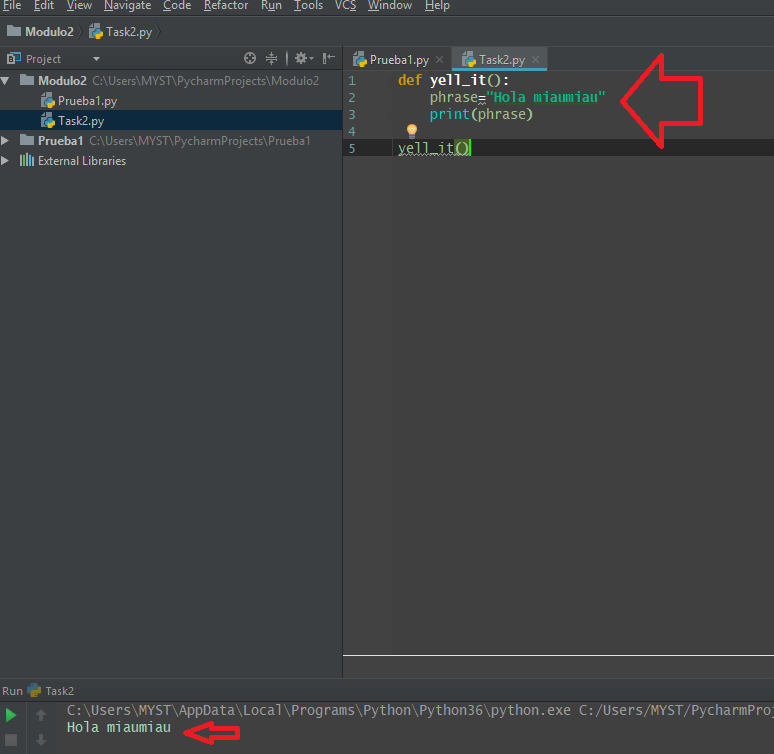
print("Hello there!")

print("goodbye")

# define three\_three

def three\_three():

print(33)



**1.4 Call Function by Name**

# Concept: Calling Functions

## Calling a function by name

Call a simple function using the function name followed by parentheses. For instance, calling print is  
**print()**

# Example

# Program defines and calls the say\_hi & three\_three functions

# [ ] review and run the code

def say\_hi():

print("Hello there!")

print("goodbye")

# end of indentation ends the function

# define three\_three

def three\_three():

print(33)

# calling the functions

say\_hi()

print()

three\_three()

# Task 2

## Define and call a simple function   yell\_it()

### yell\_it()   prints the phrase with "!" concatenated to the end

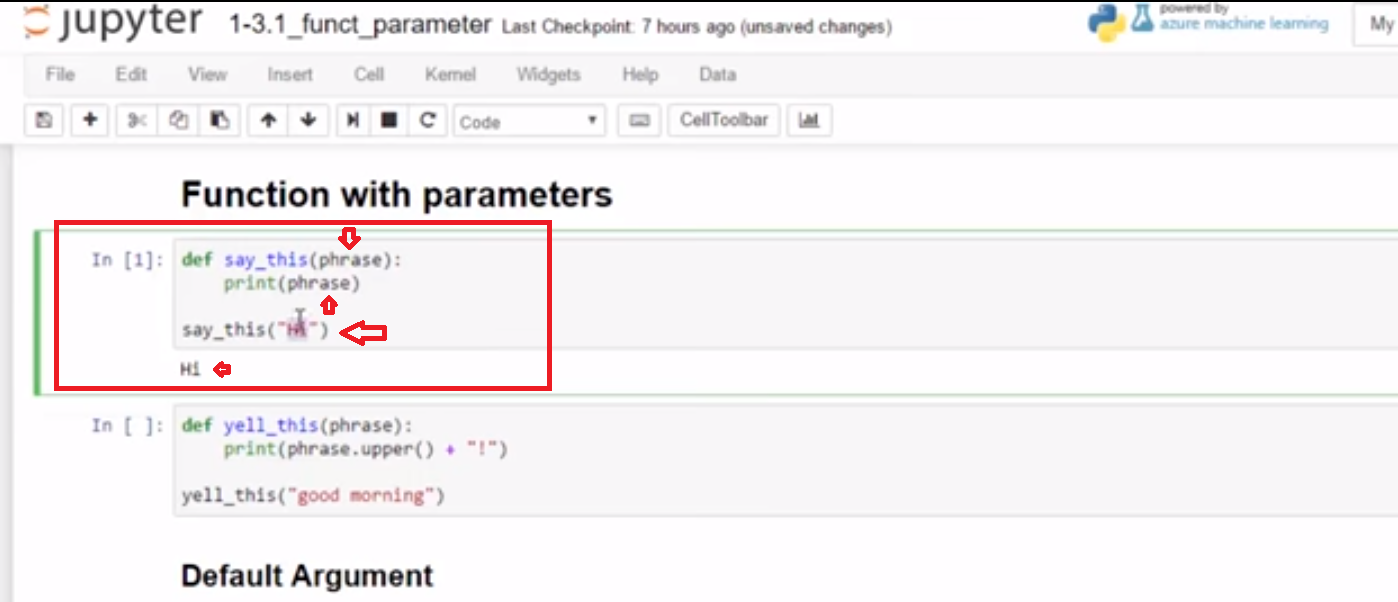
* takes no arguments
* indented function code does the following
  + define a variable for called **phrase** and initialize with a short phrase
  + prints **phrase** as all upper-case letters followed by "!"
* call   yell\_it   at the bottom of the cell after the function  **def** (**Tip:** no indentation should be used)

#[ ] define (def) a simple function called yell\_it() and call the function

**1.5 Function Parameters**

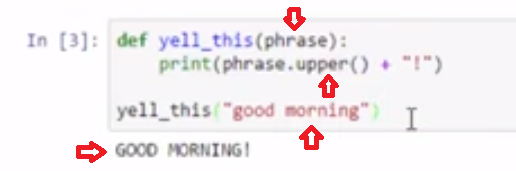
**Video:** **FunctionParametersV3.mp4**

**Function parameters allow data to be passed to and used by functions.** This example uses a parameter in the function definition. **Say\_this has phrase inside the parentheses and that is the name of a variable. That variable gets used inside the function code, in this case, in a print statement that is indented inside of the function. So to call that function, we just call it with the argument, which in this case is a string, HI.** **And there we go with the string getting passed into the variable in the parameter list and then that gets used to print.**

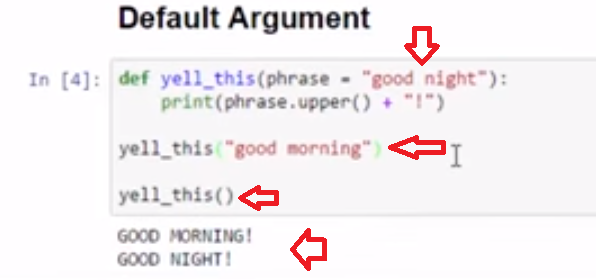


And we can change that. And we see that it will continue to update with whatever value gets passed as an argument. So the parameter can be used within the function.

**I have a very similar function called yell\_this. And it does exactly what say\_this says, except that it uses a string method to make the phrase into upper case and it uses string addition to put a exclamation point at the end.** So let's call with good morning and we see that this works as the example above.



But what if we wanted a default value, so that we can call this without any arguments? **So I want to call yell\_this just with parenthesis and no argument inside. How I would do that is when I define my parameter, I would assign it a value. So I just assigned it a default value of good night and I can still call it with an argument like good morning and I expect it to act as it did before and it'll just ignore my default value. But if I call it with no value, it will use the default value.** So let's run that code. And we see that in the first instance of the call good morning, the argument was passed and the second with no values, it uses the default.



So now, we know how to define functions with arguments and we know how to give those arguments default values.

Concept: Function Parameters

Functions that have Parameters

**print()** and **type()** are examples of built-in functions that have **Parameters** defined

**type()** has a parameter for a **Python Object** and sends back the *type* of the object

an **Argument** is a value given for a parameter when calling a function

* **type** is called providing an **Argument** - in this case the string *"Hello"*

type("Hello")

Defining Function Parameters

* Parameters are defined inside of the parenthesis as part of a function **def** statement
* Parameters are typically copies of objects that are available for use in function code

def say\_this(phrase):

print(phrase)

Function can have default Arguments

* Default Arguments are used if no argument is supplied
* Default arguments are assigned when creating the parameter list

def say\_this(phrase = "Hi"):

print(phrase)

Example

# yell\_this() yells the string Argument provided

def yell\_this(phrase):

print(phrase.upper() + "!")

# call function with a string

yell\_this("It is time to save the notebook")

# use a default argument

def say\_this(phrase = "Hi"):

print(phrase)

say\_this()

say\_this("Bye")

Task 3

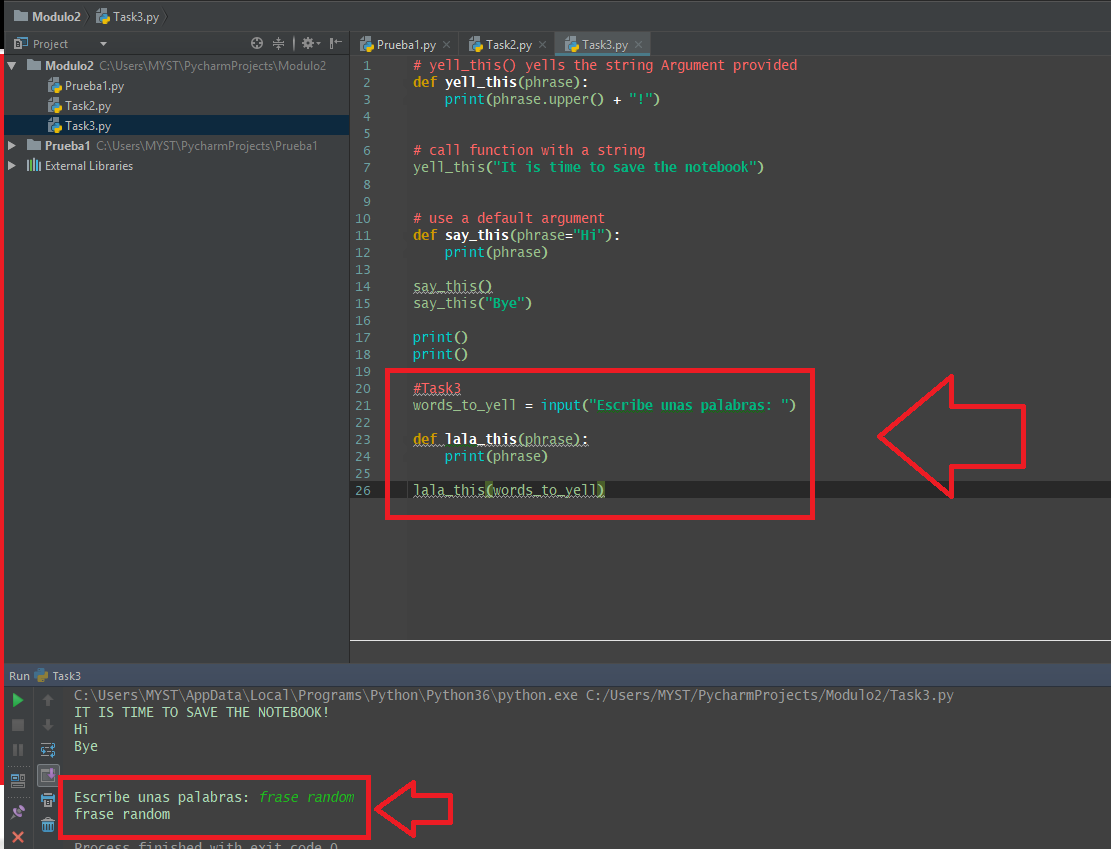
Define yell\_this() and call with variable argument

* define variable   **words\_to\_yell**   as a string gathered from user  input()
* Call  **yell\_this()**  with   **words\_to\_yell**  as argument
* get user input() for the string words\_to\_yell

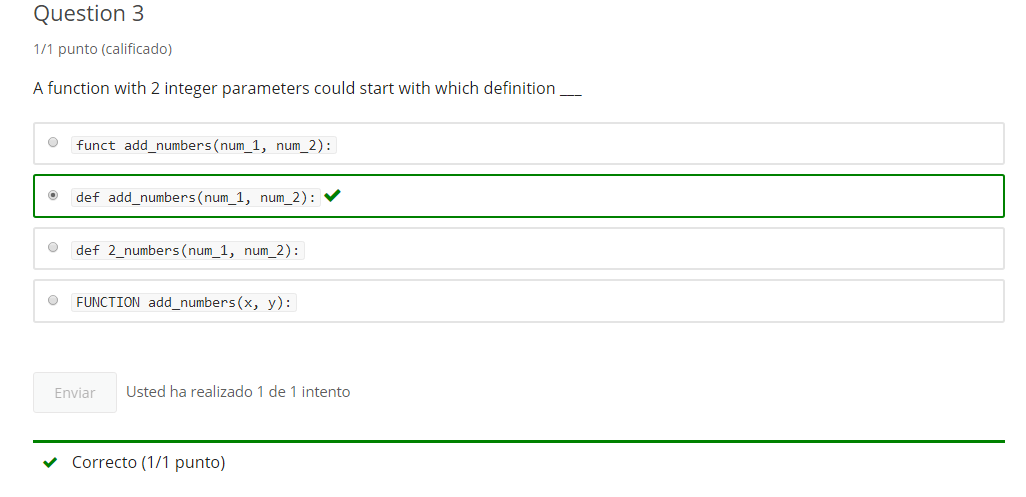
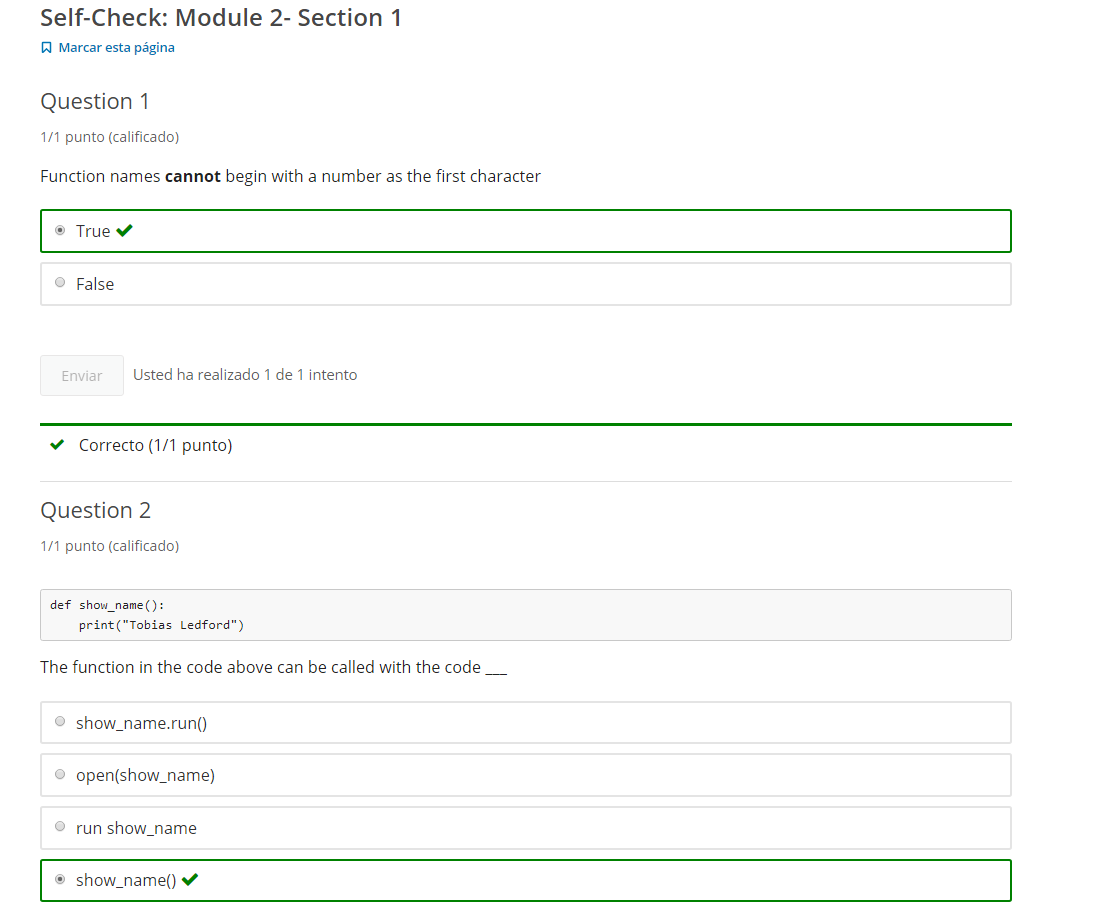
# [ ] define yell\_this()

# [ ] get user input in variable words\_to\_yell

# [ ] call yell\_this function with words\_to\_yell as argument



**1.6 Self-Check: Module 2-Section 1**





**2. Function return & multi-parameters**

**2.1 Intro Python**

Jupyter Notebook: MOD02\_1-3.2\_Intro\_Python.ipynb

Functions Arguments & Parameters

* Creating a simple Function with a parameter
* **Exploring Functions with return values**
* **Creating Functions with multiple parameters**
* Sequence in python

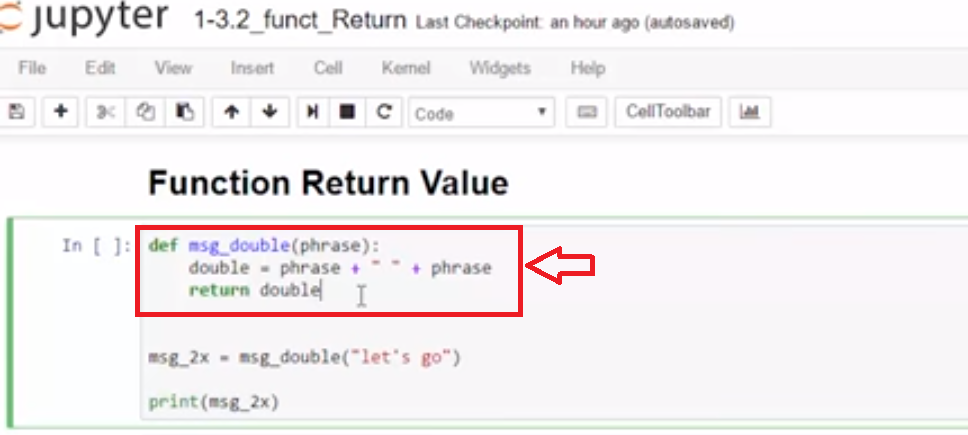
Student will be able to

* **create functions with a return value**
* **create functions with multiple parameters**

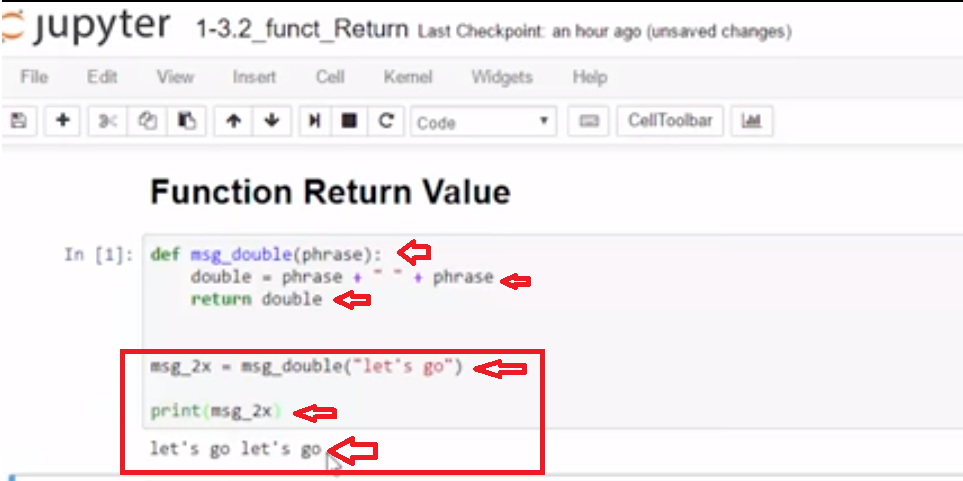
**2.2 Call Function With Return**

**Video:** **CallFunctionWithReturnV4.mp4**

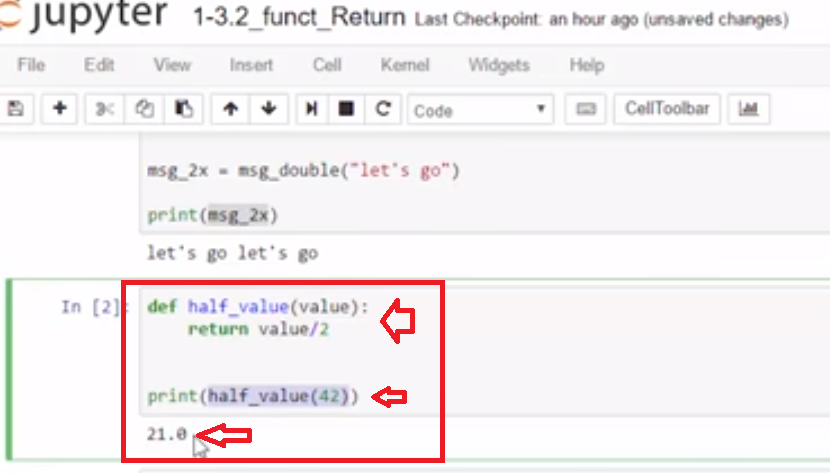
**Functions can not only take in values with arguments. They can also return, using the return keyword within the body of the function.** In our first example of a function call with a return value, we're gonna look at msg\_double. In the body of the msg\_double function in the code we see return keyword. **The return keyword is always followed by an object.** In this case, it is the variable called double. **The return value could also be followed by a literal string, or a number, or any object.**



What happens when you call a function with a return value? Here we call msg\_double with the argument let's go in the quotations, a string. The parameter phrase gets replaced with the value let's go in quotation. The variable double, Adds let's go with a space to itself. Double gets returned to replace this function call. **When you call a function with a return value, the return value replaces the function call.** Let's run the code. We see that the message let's go let's go is printed. **This is the value let's go let's go, because this function call is replaced by the return value. And we captured that return value in a variable, and printed it.**



Let's look at a slightly lighter example. This has taken away all of the variables used in the body of the function, and we just use the parameter value. The function half\_value divides the value by two. So when we call half\_value, we expect that it will be replaced by the return value. 42 is sent as the argument and is used in the parameter value, and divided by 2, so we're gonna get 21. Let's check and see. So that return value replaced the function call. And we printed it, and we see that it has a float value of 21. The keyword return allows a function to send back a value to the calling function.



Concept: Function with Return Value

Calling a function with a return value

* **type()** returns an object type
* **type()** can be called with a float the return value can be stored in a variable

object\_type = type(2.33)

creating a function with a return value

* **return** keyword in a function *returns* a value after *exiting* the function

def msg\_double(phrase):

double = phrase + " " + phrase

return double

Example

review and run the code

# Message double returns the string Argument doubled

def msg\_double(phrase):

double = phrase + " " + phrase

return double

# save return value in variable

msg\_2x = msg\_double("let's go")

print(msg\_2x)

# example of functions with return values used in functions

def msg\_double(phrase):

double = phrase + " " + phrase

return double

# prints the returned object

print(msg\_double("Save Now!"))

# echo the type of the returned object

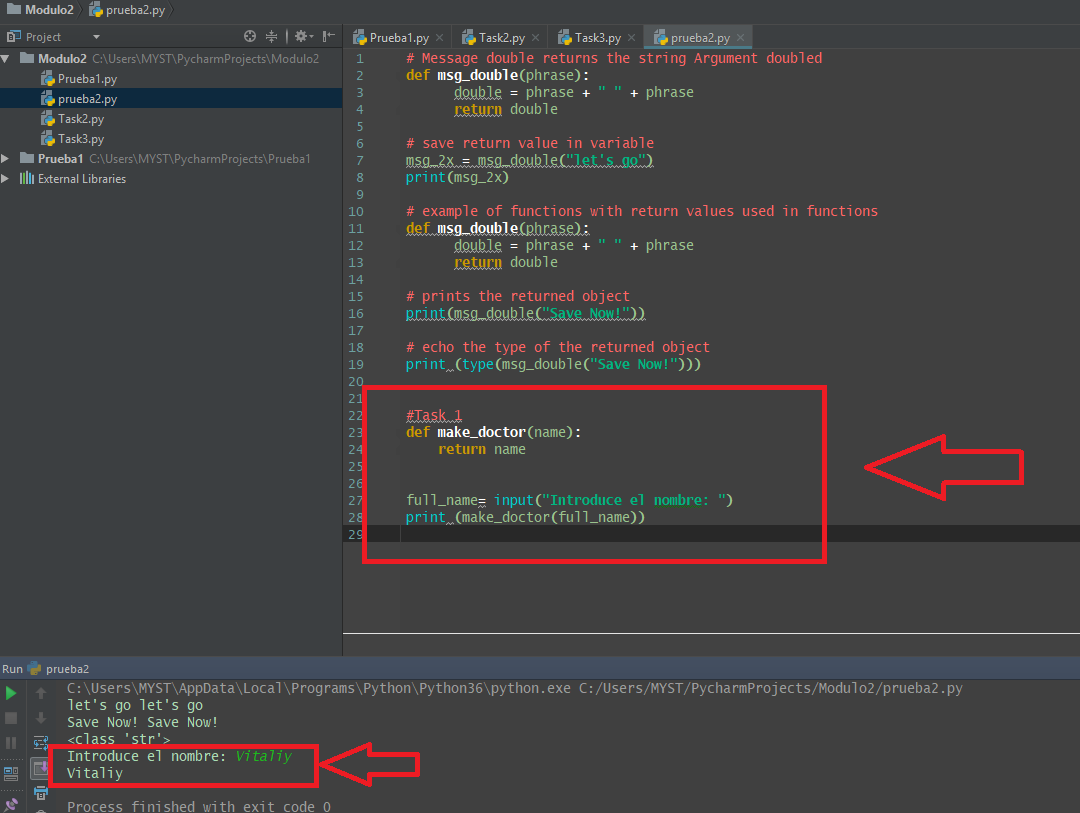
type(msg\_double("Save Now!"))

**Task 1**

Doctor: a function that adds the "Doctor" title to a name

* Define function make\_doctor()  that takes a parameter name
* get user **input** for variable **full\_name**
* call the function using full\_name   as argument
* print the return value

# create and call make\_doctor() with full\_name argument from user input - then print the return value

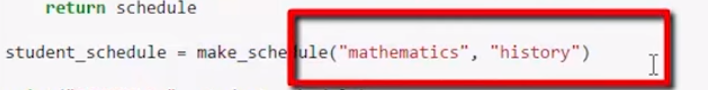


**2.3 Call Function with multi-parameters**

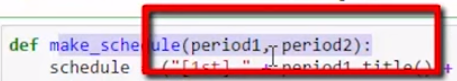
**Video:** **CallFunctionWithMulti-ParametersV5.mp4**

In order to create functions that take arguments like we saw with the print and the type function, we're going to have to define some parameters**. Sometimes we're going to need to make functions with multiple parameters. To call a function with multiple parameters, you need multiple arguments.**

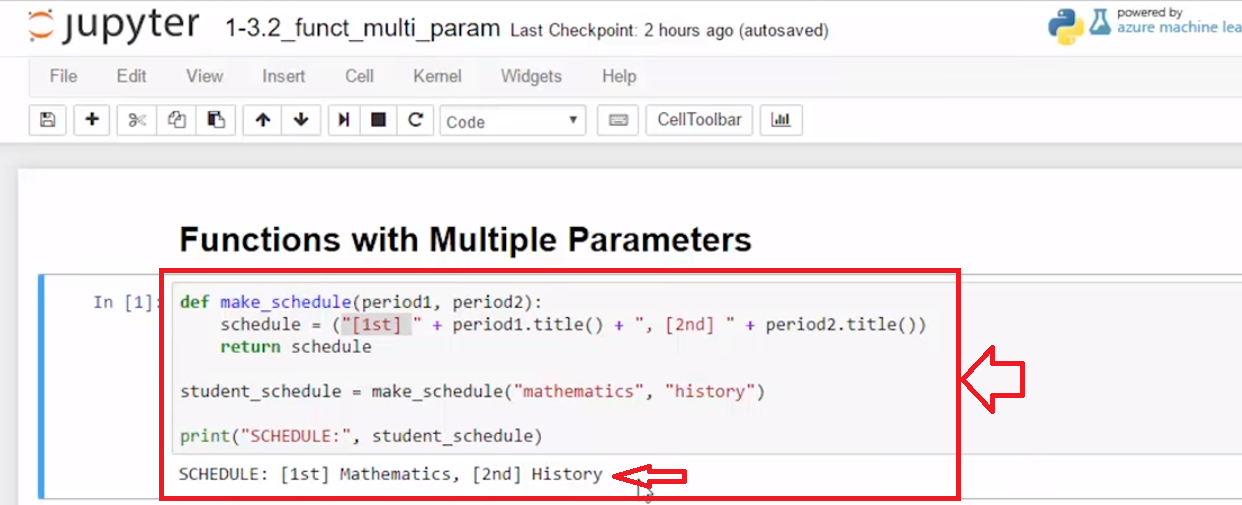
**Here we're going to call, make\_schedule with two arguments. These are strings separated by a comma.**



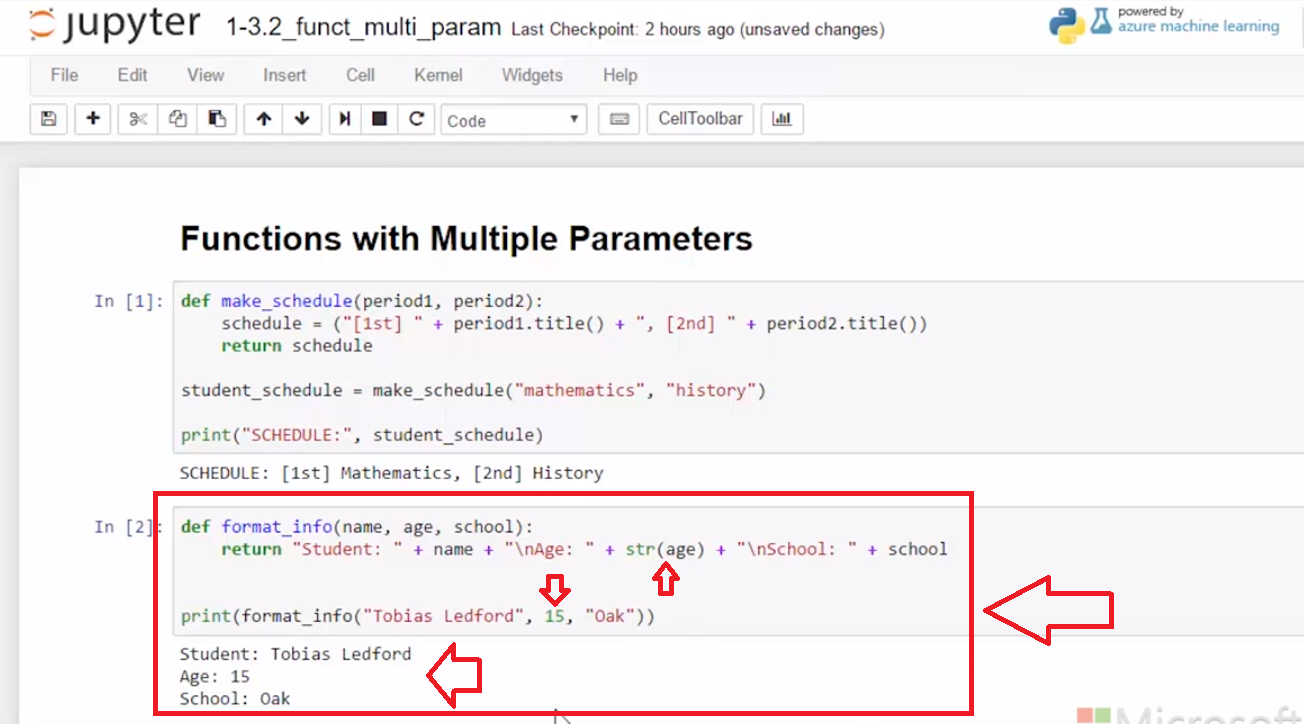
**To define the function, make\_schedule with two parameters as well inside the parentheses of the function definition, we separate the function names also with a comma.**



**Then when we call that function, the first argument, mathematics the string will be assigned to the parameter period1. And likewise history will be assigned to period2**. In the body of the function, we can do some string concatenation. And then we can return that in this variable schedule. Below **we print out that return value with a little label SCHEDULE.** Let's run the code. Here we see that we've added the first period and then that first argument mathematics, and second period, the second argument history.



**In this example, format\_info,** **we're going to send three parameters. So, here we call with the arguments a name which is a string, and an age which is a number integer and a school which is also a string.** Here the return statement is made up of a string concatenation versus a variable as we did above. Let's run that code.



And here, we can see because we use formatting characters, that we get a nice output here and we can make functions with multiple parameters by using the commas to separate the parameter names.

Example

review and run the code

def make\_schedule(period1, period2):

schedule = ("[1st] " + period1.title() + ", [2nd] " + period2.title())

return schedule

student\_schedule = make\_schedule("mathematics", "history")

print("SCHEDULE:", student\_schedule)

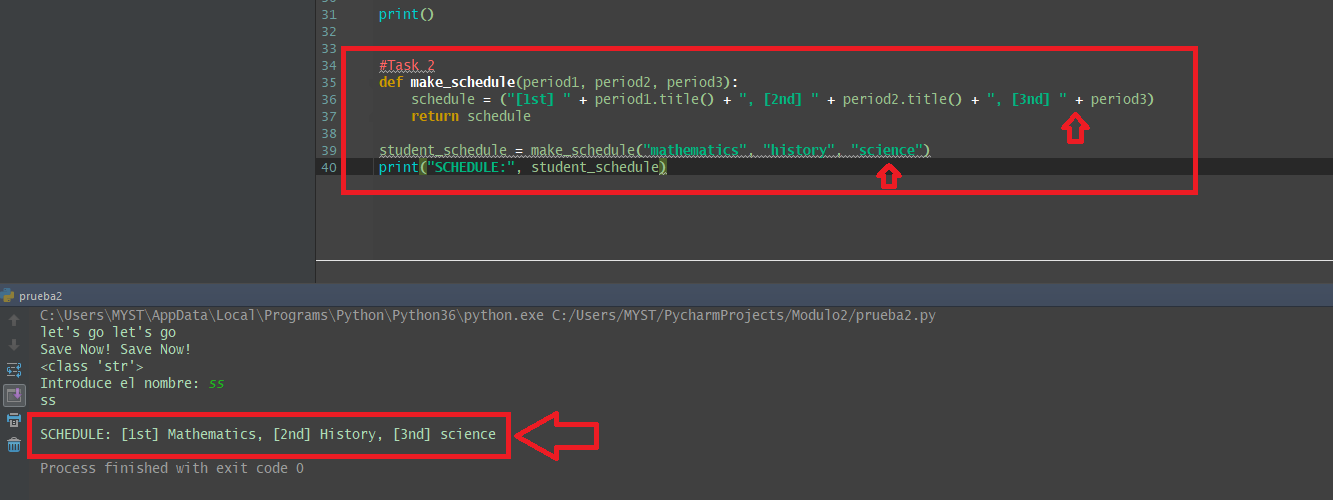
Task 2

Define make\_schedule() adding a 3rd period to

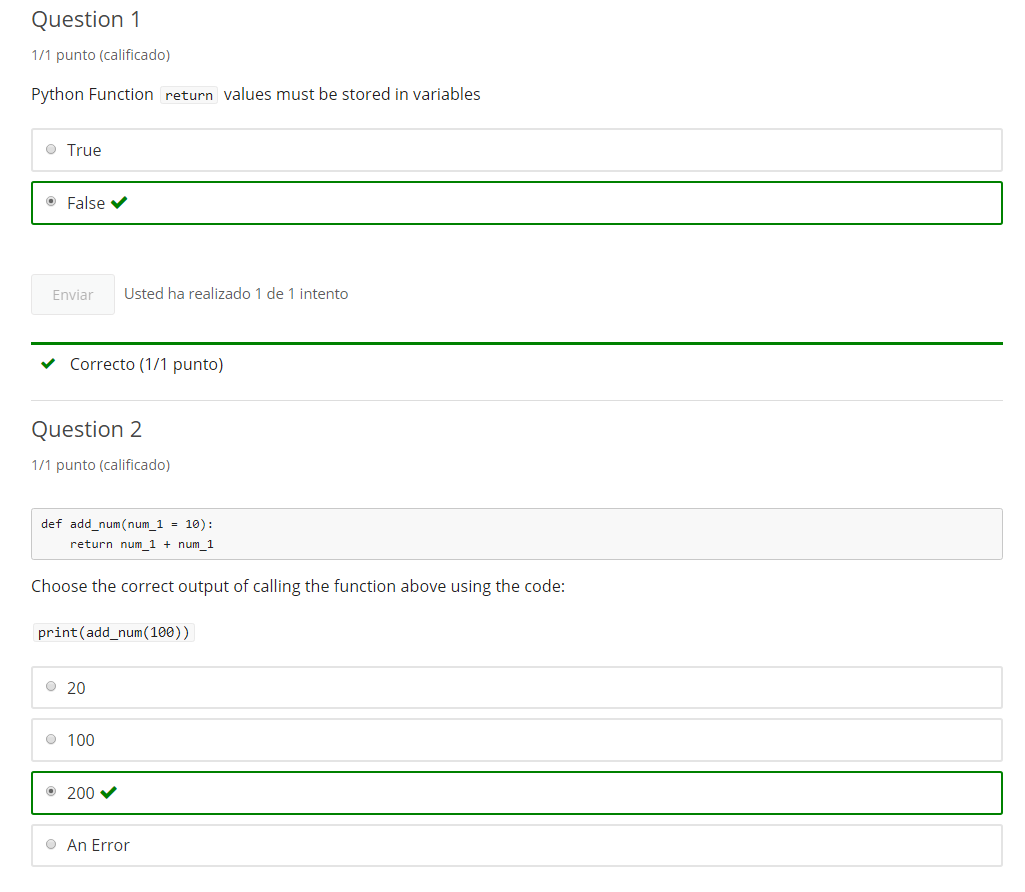
* Start with the above example code
* add a parameter period\_3
* update function code to add period\_3 to the schedule
* call **student\_schedule()** with an additional argument such as 'science'
* print the schedule

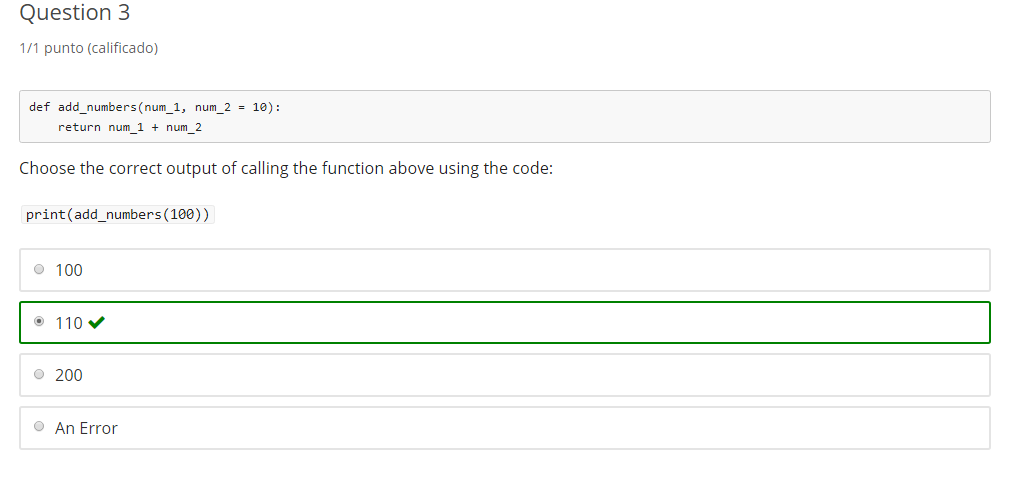
# [ ] add a 3rd period parameter to make\_schedule

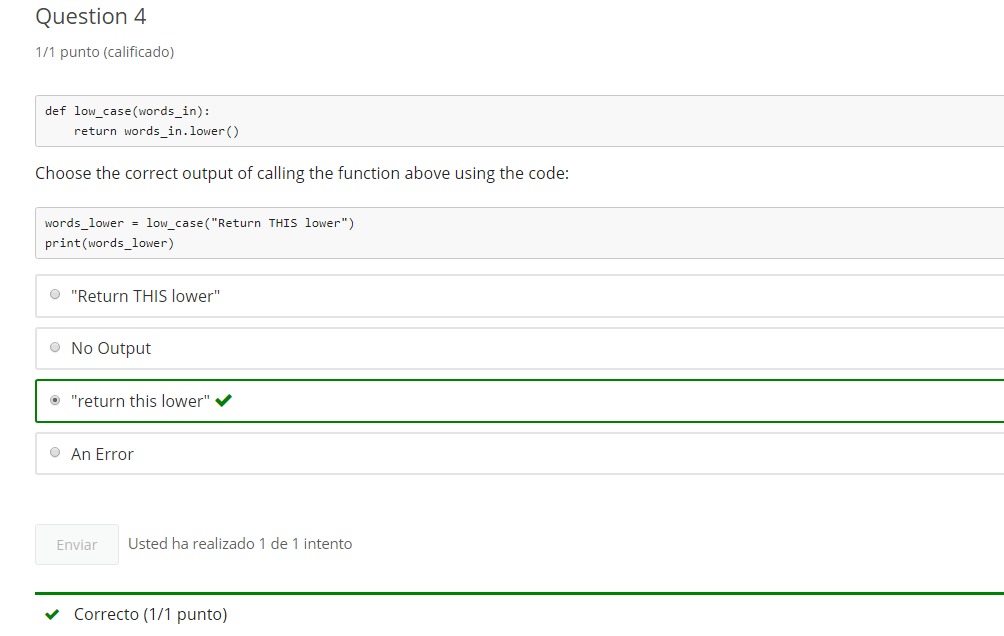
# [ ] Optional - print a schedule for 6 classes (Tip: perhaps let the function make this easy)



**2.4 Self-Check: Module 2-Section 2**







**3. Sequence**

**3.1 Intro Python**

Jupyter Notebook: MOD02\_1-3.3\_Intro\_Python.ipynb

Functions Arguments & Parameters

* Creating a simple Function with parameters
* Exploring Functions with return values
* Creating Functions with multiple parameters
* **Sequence in python**

Student will be able to

* **Use knowledge of sequence in coding tasks**
* **Use coding best practices**

**3.2 Considering sequence**

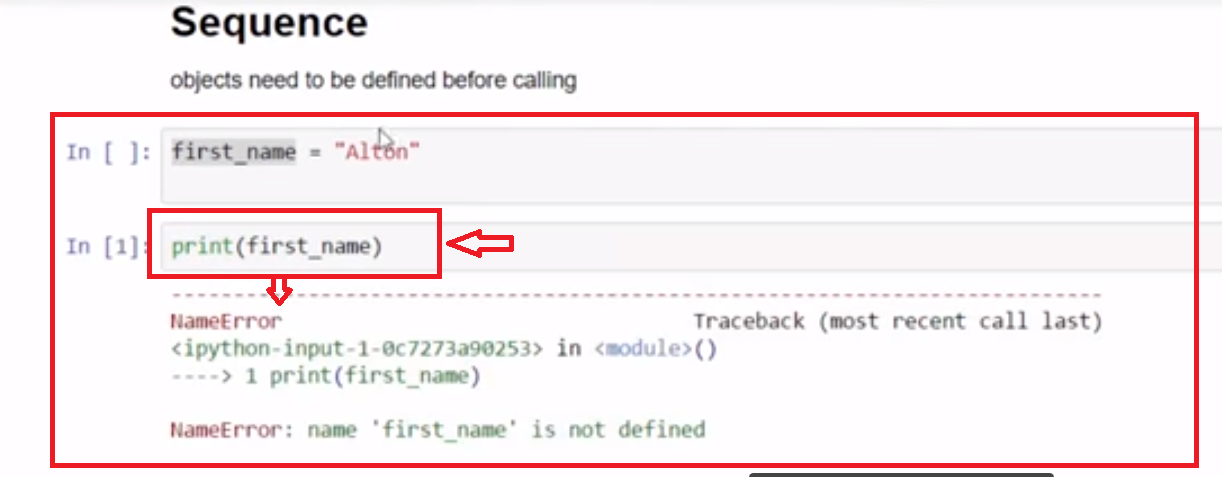
**Video:** **ConsideringSequenceV6.mp4**

# Concept: Sequence

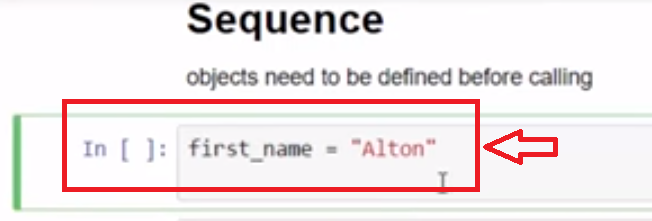
In programming, **sequence** refers to the order that code is processed. Objects in Python, such as variables and functions, are not available until they have been processed.

**Processing sequence flows from the top of a page of code to the bottom. This often means that Function definitions are placed at the beginning of a page of code.**

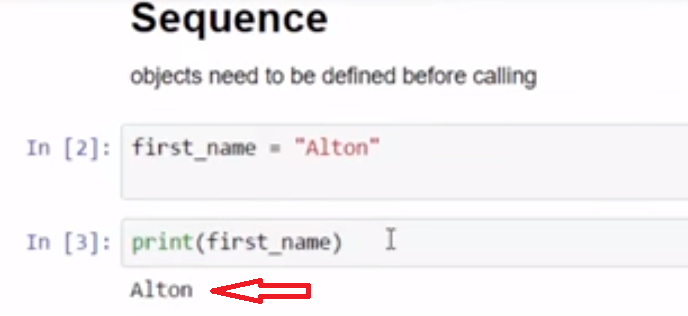
**Sequence is the order that code runs. An object needs to be defined before calling.** In this example, we have a freshly opened notebook. **first\_name is a variable that we want to to assign a name "Alton" and we're just gonna call it with a print function. I get a name error, says 'first\_name' is not defined.**



**I need to run this first cell in order to assign a value to first name and instantiate the variable.**



**Now that it's been run, I can run the print statement and print Alton**.



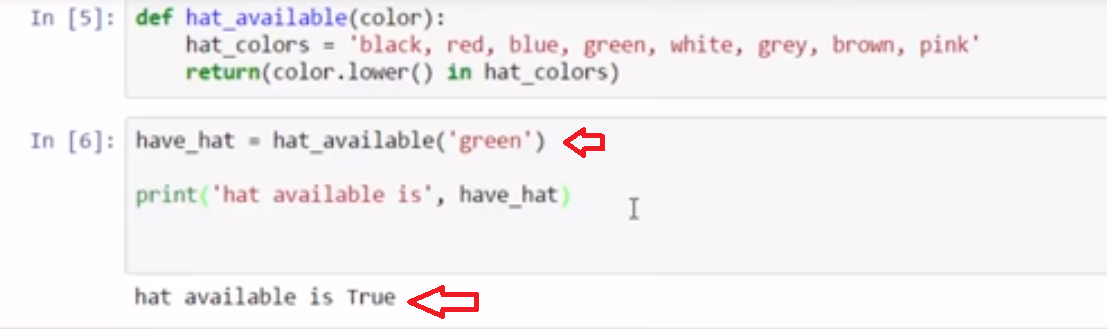
**In the second example I have defined a function hat\_available, which takes a single parameter color.** If I run this have\_hat which calls hat\_available, I run into the same problem. 'hat available' is not defined.



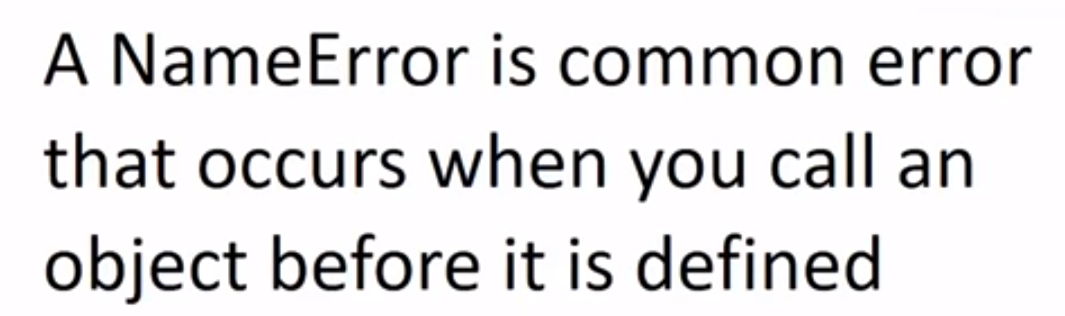
**I need to first, run the code that defines hat\_available.**



**Once that is run, I can call hat\_available.**



**A NameError is a common error that occurs when you call something before it's defined. In other words, you're calling it out of sequence.**



In the sample below, the function **hat\_color** cannot be accessed since it is initialized after it is called at the bottom of the code.

have\_hat = hat\_available('green')

print('hat available is', have\_hat)

def hat\_available(color):

hat\_colors = 'black, red, blue, green, white, grey, brown, pink'

return(color.lower() in hat\_colors)

This results in an error - the code flows from top to bottom is in the incorrect **sequence**

NameError: name 'hat\_available' is not defined

In the statement **have\_hat = hat\_available('green')** the function **hat\_available()** needs to be called after the function has been defined

**Note:** an argument or variable is said to be **hard coded** when assigned a literal or constant value.  
It is a good habit to avoid creating hard coded values in functions, such as  
hat\_colors = 'black, red, blue, green, white, grey, brown, pink'

**Example**

# review and run code - note: fix error in the following "tasks" section

have\_hat = hat\_available('green')

print('hat available is', have\_hat)

def hat\_available(color):

hat\_colors = 'black, red, blue, green, white, grey, brown, pink'

# return Boolean

return(color.lower() in hat\_colors)

**Task 1**

Change the Sequence to fix the NameError

* [ ] fix the code **sequence** so the **hat\_available()** function is available when called and the code runs without error

# [ ] fix the sequence of the code to remove the NameError

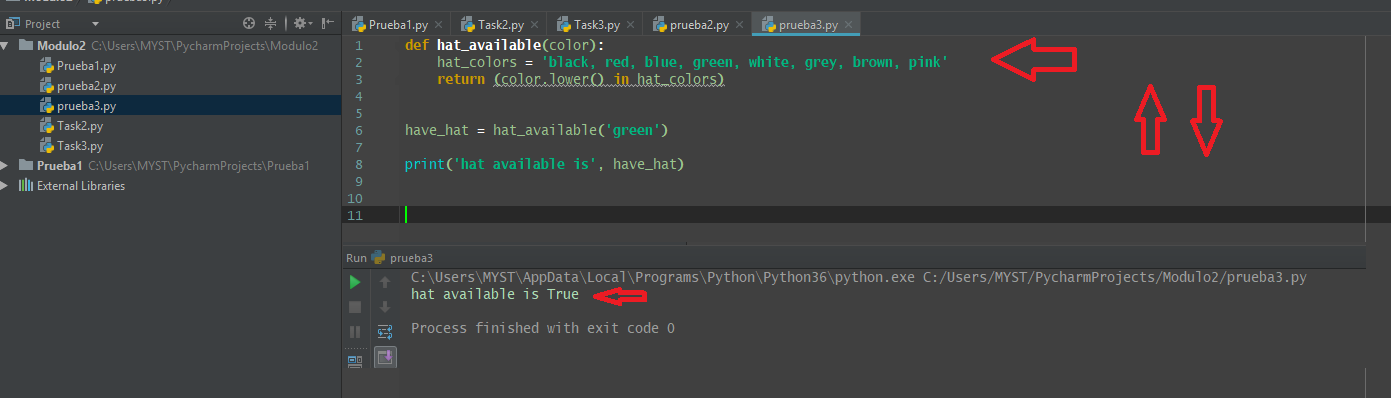
have\_hat = hat\_available('green')

print('hat available is', have\_hat)

def hat\_available(color):

hat\_colors = 'black, red, blue, green, white, grey, brown, pink'

return(color.lower() in hat\_colors)



**3.3 Avoid Hard Coded Values**

**Video:** **AvoidHardCodedValuesV7.mp4**

# Concept: Don't "hard" code

## Programming Style Tip: Avoid Hard-Coding

### "Hard-coding" is placing data values directly into code

An example of hard-coding from above is **have\_hat = hat\_available('green')** where the argument 'green' is hard-coded

A programming best practice is to **avoid hard-coding values when possible**

* Use variables and verse hard-coded (example: color = "green")
* Often preferable to use input such as a configuration file (advanced topic) or user input.

These practices allow changing the data without disturbing the main code and makes code more reusable.

# Task 2

## Program: bird\_available

The program should ask for user to "input a bird name to check for availability" and print a statement informing of availability

Here are some tips in solving the Bird Available problem. To make this task easier to understand, let's just run a sample of the completed program. So, we see that we need to create a label that ask for input about the type of bird to search. I entered chicken, because chicken is not on the list of the available bird types in this string. So let's go ahead and enter that and we see that chicken available is false. So you should test something like that, that is not in the list to make sure that it works as false. Now, let's look for something that is in the string, and so I'm gonna type crow. And crow available is true, so we know that works. What would happen, though, if I were to type crow so it was not the same case? So I capitalize crow, and it's still found. So that means, when we're doing our comparison for the string from the input, that we're gonna have to do something to make sure that the cases don't really matter when we're doing the comparison.

### create this program with a Boolean function bird\_available()

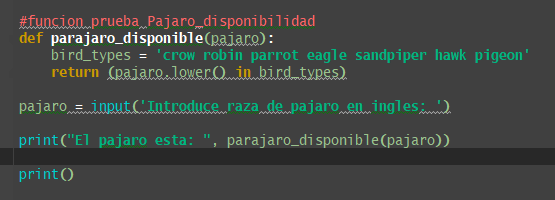
* has parameter that takes the name of a type of bird
* for this exercise the variable bird\_types = 'crow robin parrot eagle sandpiper hawk pigeon'
* return True or False (we are making a Boolean function)
* call the function using the name of a bird type from user input
* print a sentence that indicates the availability of the type of bird checked

# [ ] create function bird\_available

# [ ] user input

# [ ] call bird\_available

# [ ] print availability status



Se usa **.lower()** para que a la hora de buscar si escribo en mayuscula la transforme en minuscula y encuentre el pajaro.

# Task 3

## Fix The Error

# define function how\_many

how\_many():

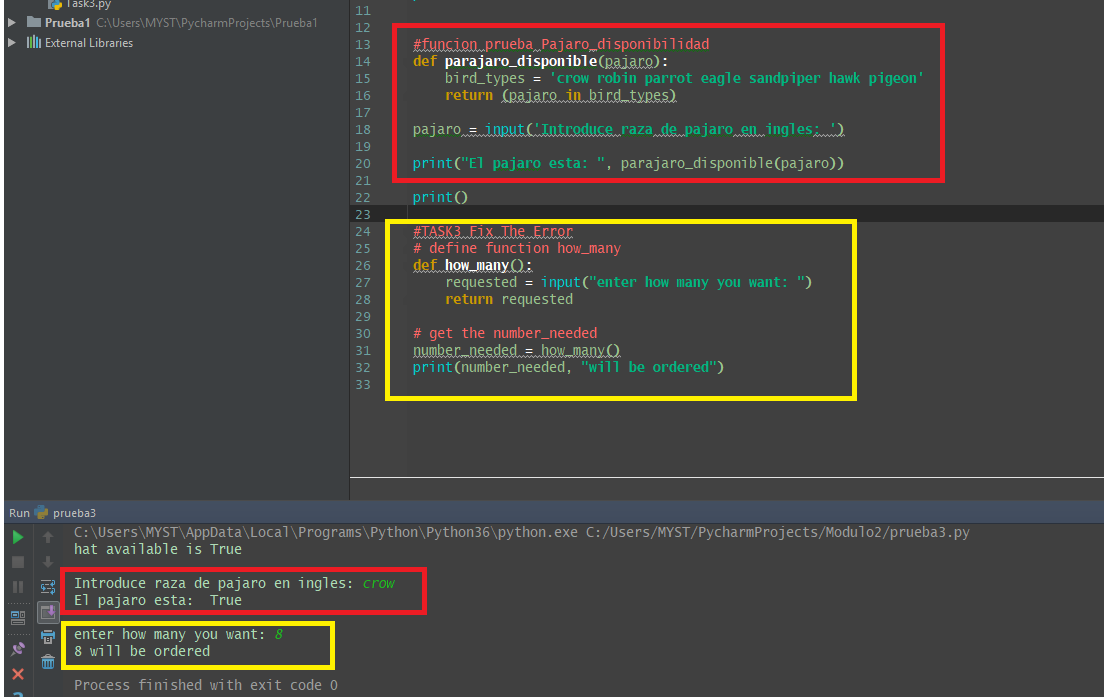
requested = input("enter how many you want: ")

return requested

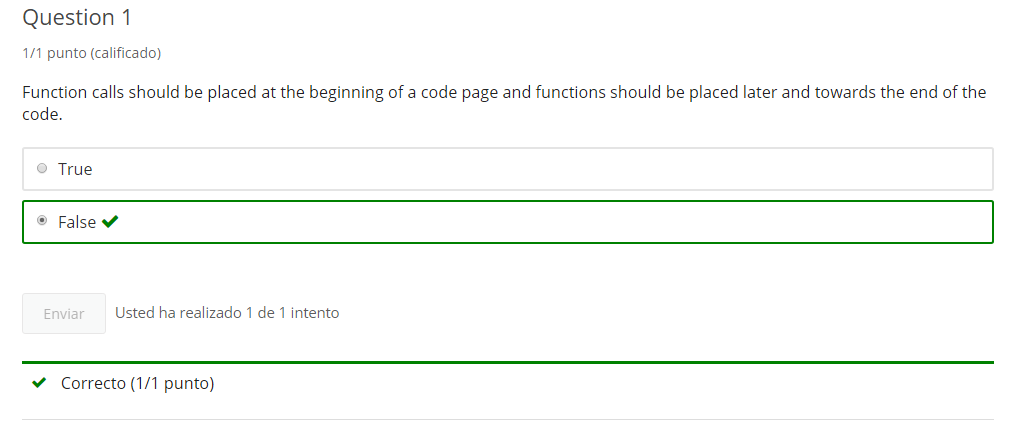
# get the number\_needed

number\_needed = how\_many()

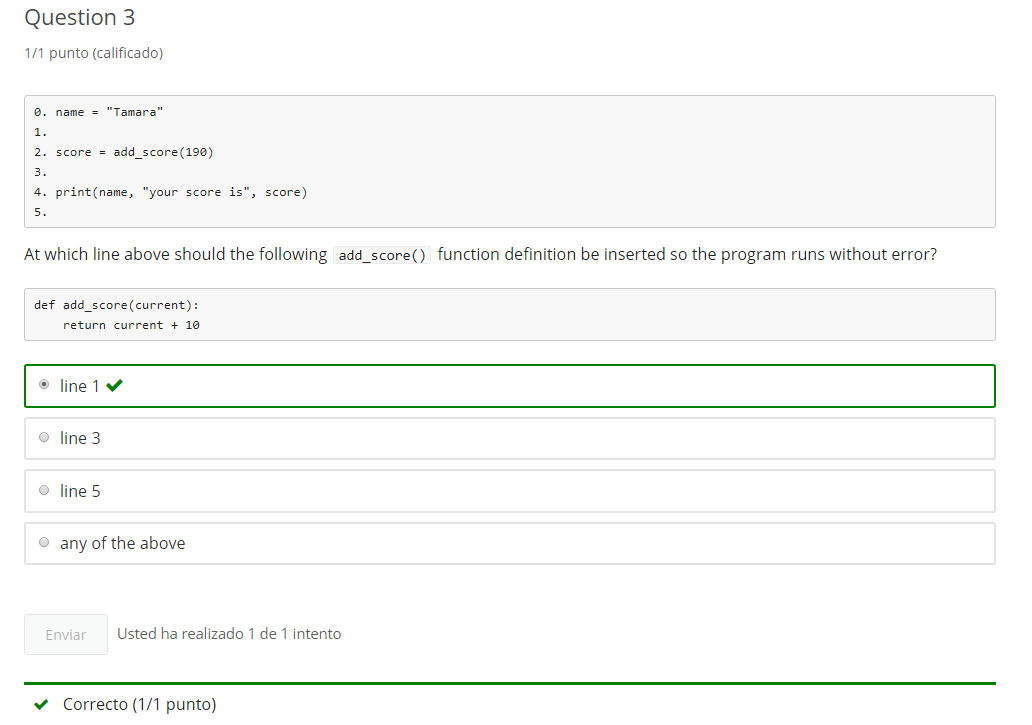
print(number\_needed, "will be ordered")

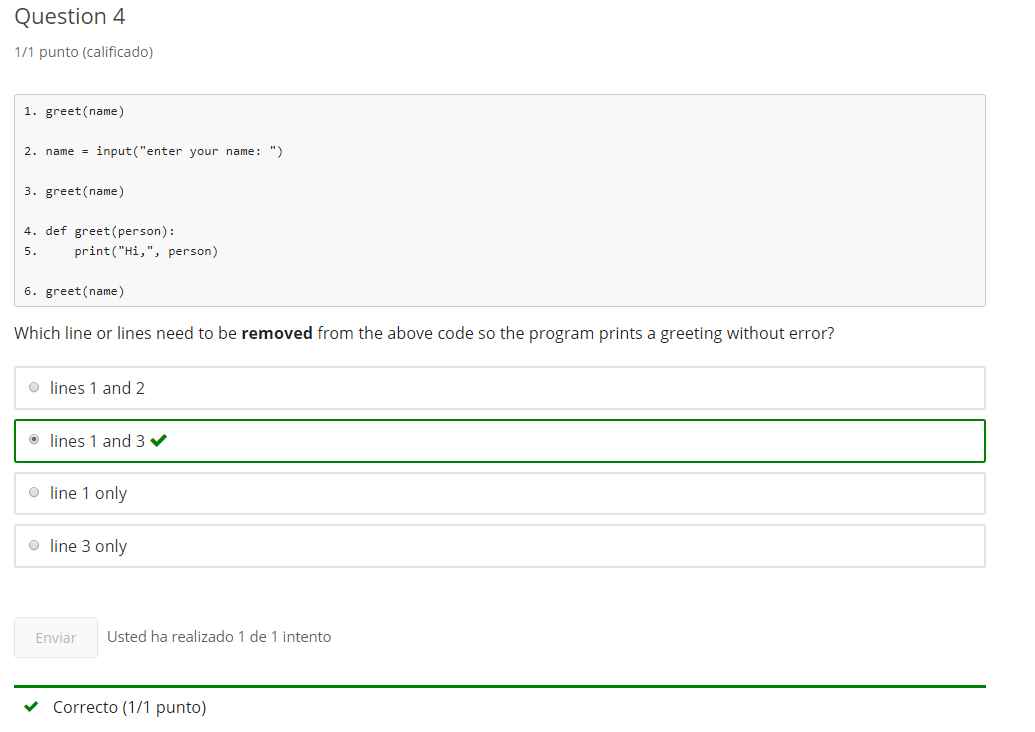


**3.4 Self-Check: Module 2-Section 3**









**4. Function Practice**

**4.1 Intro Python Practice - notebook 3**

# 1-3 Intro Python Practice

optional practice

Jupyter Notebook: MOD02Practice\_1-3\_IntroPy.ipynb

## Functions Arguments & Parameters

# Student will be able to

* **create functions with a parameter**
* **create functions with a return value**
* **create functions with multiple parameters**
* **use knowledge of sequence in coding tasks**
* **use coding best practices**

# Tasks

# [ ] define and call a function short\_rhyme() that prints a 2 line rhyme

# [ ] define (def) a simple function: title\_it() and call the function

# - has a string parameter: msg

# - prints msg in Title Case

# [ ] get user input with prompt "what is the title?"

# [ ] call title\_it() using input for the string argument

# [ ] define title\_it\_rtn() which returns a titled string instead of printing

# [ ] call title\_it\_rtn() using input for the string argument and print the result

## Program: bookstore()

create and test bookstore()

* **bookstore() takes 2 string arguments: book & price**
* **bookstore returns a string in sentence form**
* **bookstore() should call title\_it\_rtn()** with book parameter
* **gather input for book\_entry and price\_entry to use in calling bookstore()**
* **print the return value of bookstore()**

example of output:**Title: The Adventures Of Sherlock Holmes, costs $12.99**

# [ ] create, call and test bookstore() function

### Fix the error

def make\_greeting(name, greeting = "Hello"):

return (greeting + " " + name + "!")

# get name and greeting, send to make\_greeting

print(make\_greeting(get\_name(), get\_greeting()))

def get\_name():

name\_entry = input("enter a name: ")

return name\_entry

def get\_greeting():

greeting\_entry = input("enter a greeting: ")

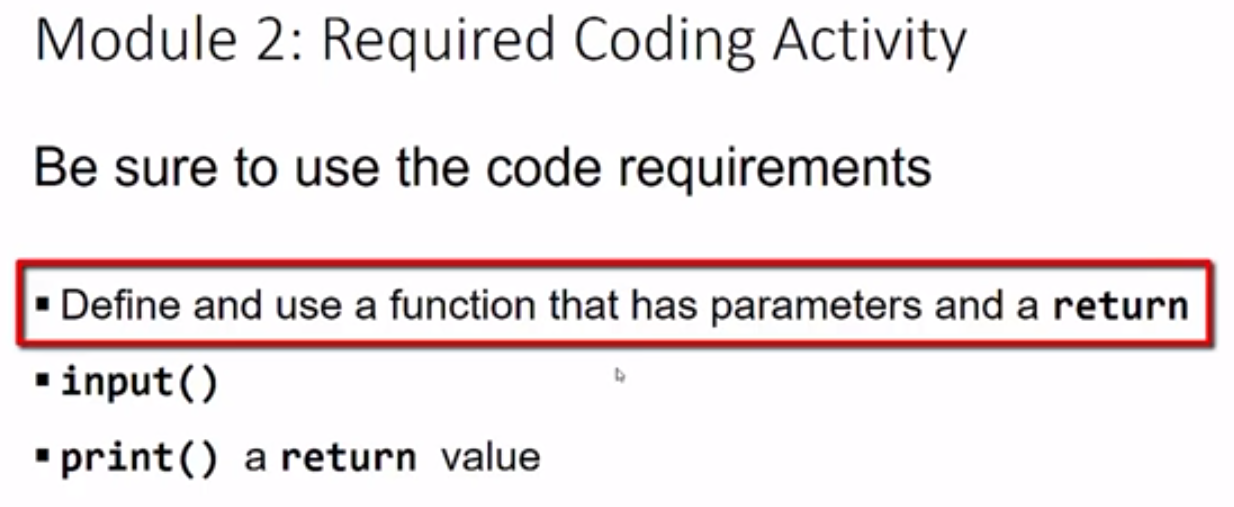
return greeting\_entry

**5. End of Mod coding assignment**

**5.1 Module 2 Required Code Description**

**Video:** **RequiredCodeDescriptionV8.mp4**

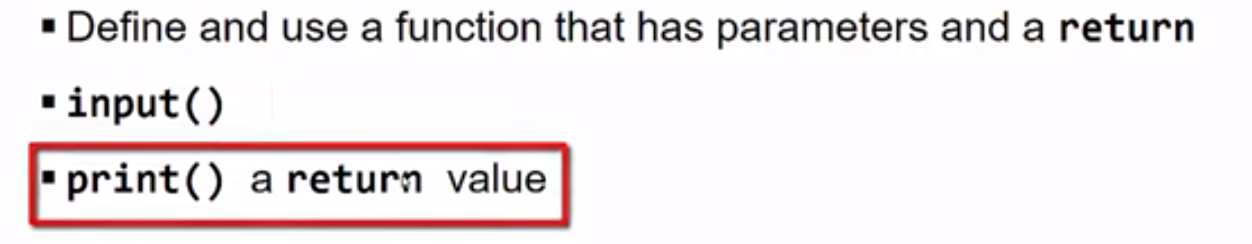
>> Let's walk through Module 2: Required Coding. And**, we're going to build a fish store function.** And it has some requirements for the code as do all of the required coding activities. So we're going to have to use specific keywords in order to get credit. So in this one it says you're going to have to define and use a function that has parameters in return. So that means we're going to use the def keyword, that **our function is going to take two arguments: It's going to be the fish name and the fish price.** And that, it's going to **return a string that has a message about the fish name and price.**



We're going to use input to get strings that we'll use as the arguments for our function,



and we're going to print the return value that comes back from the function.



So we'll have to do all of those things. Let's look at the edX course page and you can see they have those requirements there. And that, we will have to go and submit that when we're done with our submission tab. And we should do our coding in the Jupyter Notebook, or if you want to use another solution you can do that. We're going to do at the Jupyter Notebook, and I have preloaded a solution, and I'm just going to run that code here, and it ask me to enter a fish name. I'm just going to say tuna. And then, the price of the fish, I'll say 11.97. And, we see I get this nicely formatted message back. Fish type is tuna that costs 11.97 and that's what your code should do as well.



Now, let's go back to edX and paste in our code solution, and we know that from our module 1 that this is of primitive solution for submitting code, and that it will not show you the user, the indentation of the code and that's okay. So it just kind of looks like that. And then, when we submit it, we can see, you get two attempts, I've used one. So if you've missed it you've probably missed one of the requirements, such as a keyword like return. So, go ahead, and work on that solution, and submit your code. But, don't forget that we're going to have to use each of the keywords, def return, input, and print.

Module 2 Required Coding Activity

Introduction to Python Unit 1

**This Activity is intended to be completed in the jupyter notebook, Required\_Code\_MOD2\_IntroPy.ipynb and then pasted into the assessment page that follows.**

All course .ipynb Jupyter Notebooks are available from the project files download topic in Module 1, Section 1.

This is an activity based on code similar to the Jupyter Notebook **Practice\_MOD02\_1-3\_IntroPy.ipynb** which you may have completed.

| **Some Assignment Requirements** |
| --- |
| **NOTE:** This program requires a **function** be defined, created and called. The call will send values based on user input. The function call must capture a returnvalue that is used in print output. The function will have parameters and return a string and should otherwise use code syntax covered in module 2. |

Program: fishstore()

create and test fishstore()

* **fishstore() takes 2 string arguments: fish & price**
* **fishstore returns a string in sentence form**
* **gather input for fish\_entry and price\_entry to use in calling fishstore()**
* **print the return value of fishstore()**

example of output: **Fish Type: Guppy costs $1**

# [ ] create, call and test fishstore() function

# then PASTE THIS CODE into edX

Important: [How to submit code by pasting](https://courses.edx.org/courses/course-v1:Microsoft+DEV236x+1T2017/wiki/Microsoft.DEV236x.1T2017/paste-code-end-module-coding-assignments/)

**5.2 Module 2 Required Code Submission**

