**Module 1: Sequence Index**

**1. Using Jupyter**

No pongo nada, ya lo vi en el curso anterior.

**2. Index Sequences**

**2.1 Intro Python**

Jupyter Notebook: Mod1\_2-1.1\_Intro\_Python.ipynb

The link to the .ipynb Jupyter Notebook files are in the last lesson of section 0 of module 1

**Sequence: String**

* **Accessing String Characters with index**
* Accessing sub-strings with index slicing
* Iterating through Characters of a String
* More String Methods

**Student will be able to**

* **Work with String Characters by index**

**2.2 String Index**

**Video:** **AccesingSingleStringCharacterV1.mp4**

# We can Access a single character of a String using a String index address. Let's take a look at the image in the screen here and it represents a string, the word Alton. The name Alton broken down by each character in the string and below each of the characters is a number which is the index for each of the characters. So we have the first character with index 0, "A" and then the second character is index 1. Counting from 0 is a common way that computers do calculations. So it's something to be aware of and we'll see that in several pieces of code.

# 

# Let's go ahead and look at the code below and we see that we are going to use the student named "Alton" and we're going to access that student name and then follow that with square brackets with the index number inside. That's the way that we can call that the first representation of this box, that first index, and then get the first character back. So we're going to get the first character at index 0 by running that and so that will give us index 0 - "A" and index 3 will be the fourth character because we start at 0, 1, 2, 3, that's four characters and we get the "O". So let's run the code and we see as expected we bring back the characters with the indexes 0 through 4 and gives us five characters back.

# 

# In this next example, we have the student named "Jin" and that's the student name and we use that student\_name variable here and we call the first index item with the square brackets. That should return the letter J, convert that to a lowercase and see if it's "equal to" which it will not be. Then we'll go to elif statement and it will check if that is equal to lower case "j" and since we've converted it to lower that will be correct. And we'll print winner name starts with "J". So we have the two winners if it's either starts with "A" or "J" and then the else statement is not a match. So let's run that code and we say winner starts with J.

# 

# So we can change that to get the first case by changing that to "Alton" and then we see that did indeed run that first if statement

# 

# and to get to the else case we need to use something without an "A" or "J" in the first position. So I'll go to "Colette" and we see not a match.

# 

# We can also look at an example of a six letter strings "Tobias" What is the last index? in a six-letter string and so I put in six to check that and I got a index error with that says the string is out of range. We recall that we always start at zero (0) and so to count to six would be 0,1,2,3,4,5 and that would be my sixth letter and so if I run that again, I see that last letter as shows up as expected. Remember that the first character in a string has a index of 0.

# 

# Concept

## Accessing a single String Character

### addressing a string index

Strings are sequences of characters. Another common sequence type used in this course is a **list**. Sequences index items counting from 0 for the first item.

string with index for each letter

# assign string to student\_name

student\_name = "Alton"

# first character is at index 0

student\_name[0]

## Examples

# [ ] review and run example - note the first element is always index = 0

student\_name = "Alton"

print(student\_name[0], "<-- first character at index 0")

print(student\_name[1])

print(student\_name[2])

print(student\_name[3])

print(student\_name[4])

# [ ] review and run example

student\_name = "Jin"

if student\_name[0].lower() == "a":

print('Winner! Name starts with A:', student\_name)

elif student\_name[0].lower() == "j":

print('Winner! Name starts with J:', student\_name)

else:

print('Not a match, try again tomorrow:', student\_name)

# [ ] review and run ERROR example

# cannot index out of range

student\_name = "Tobias"

print(student\_name[6])

## Task 1

## Work with individual string characters

|  |
| --- |
| **Remember:** the first character in a string is at **index 0** |
|  |

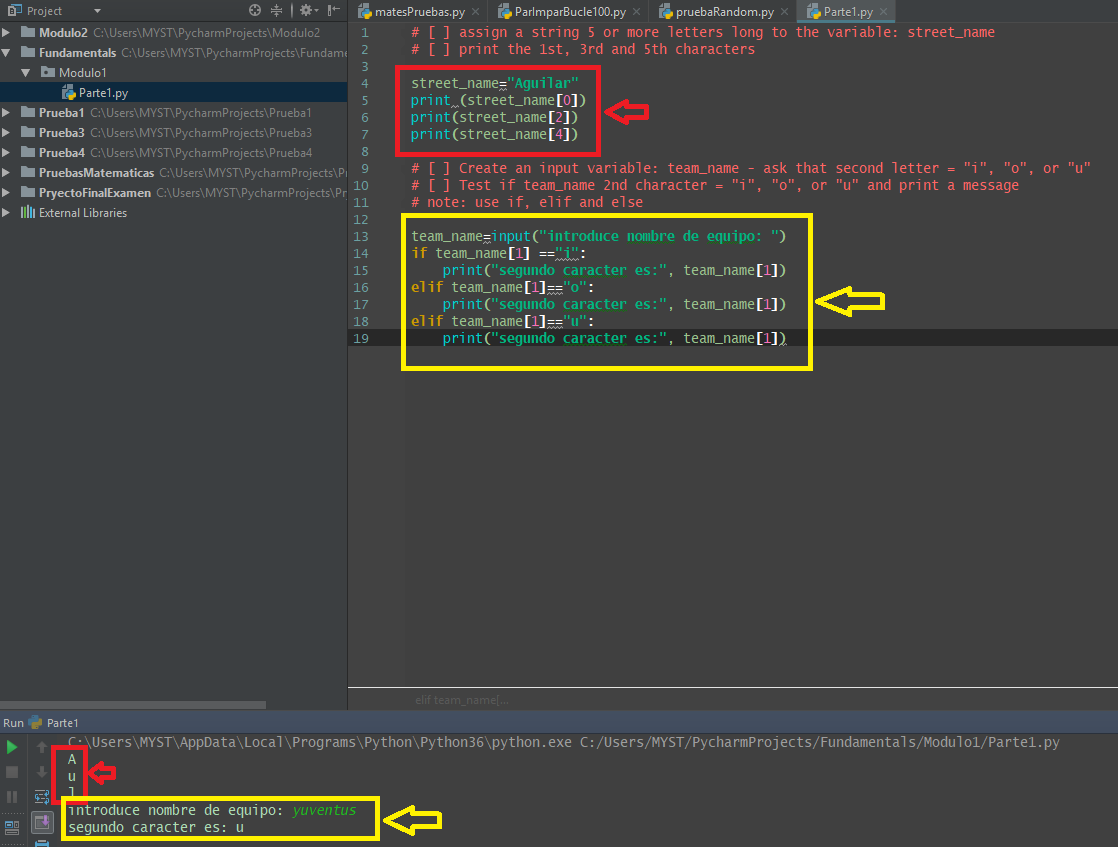
# [ ] assign a string 5 or more letters long to the variable: street\_name

# [ ] print the 1st, 3rd and 5th characters

# [ ] Create an input variable: team\_name - ask that second letter = "i", "o", or "u"

# [ ] Test if team\_name 2nd character = "i", "o", or "u" and print a message

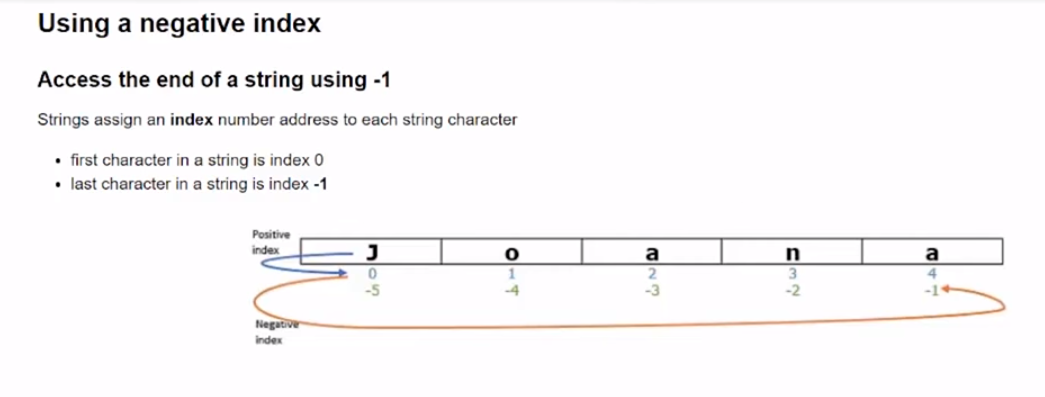
# note: use if, elif and else



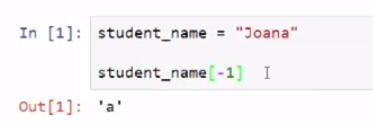
**2.3 Negative Index**

**Video:** **UsingNegativeIndexV2.mp4**

**We can address individual charactersin a string from the last character counting backwards using a negative index address.** This image on the screen will also help us visualize the indexes counting from zero and using a negative index, which start at the last character with negative one, "J" in the first position is at index zero, and the letter "A" at the last position is at index 4. Or we can use negative indexing and see that the "J" is negative five (-5) because the "A" position, the last letter is negative one (-1). So we count back from the negative two, negative three, negative four, negative five, and then we arrive on the "J" index.

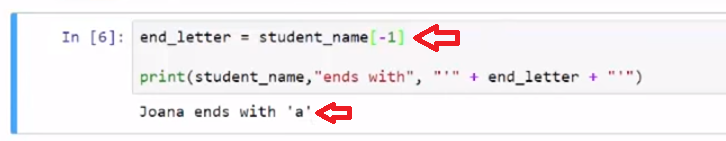


In the example, we have student\_name Joana, just like we have on the image. And we're gonna run that negative one position and we do see the "A" as we expected.

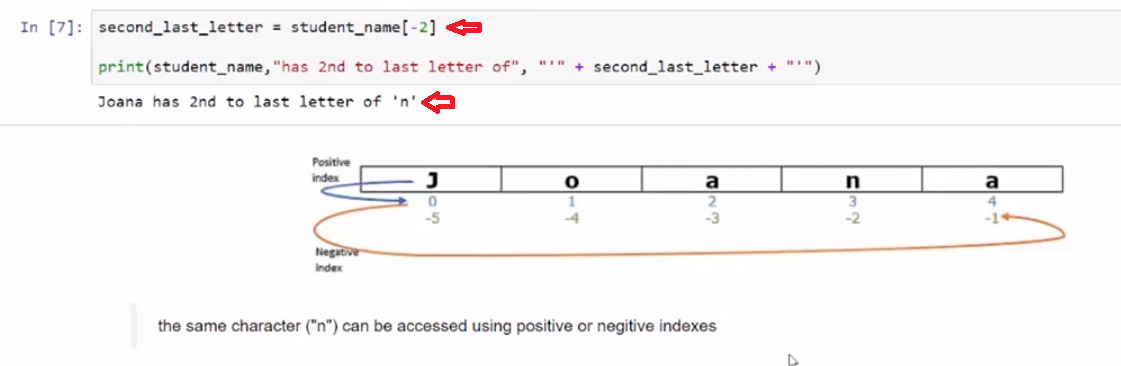


And we can run that five and we can get the "J". And just like we saw before, we can run the zero, also "J", and we can run four, and also "A", the same as if we were at negative one.

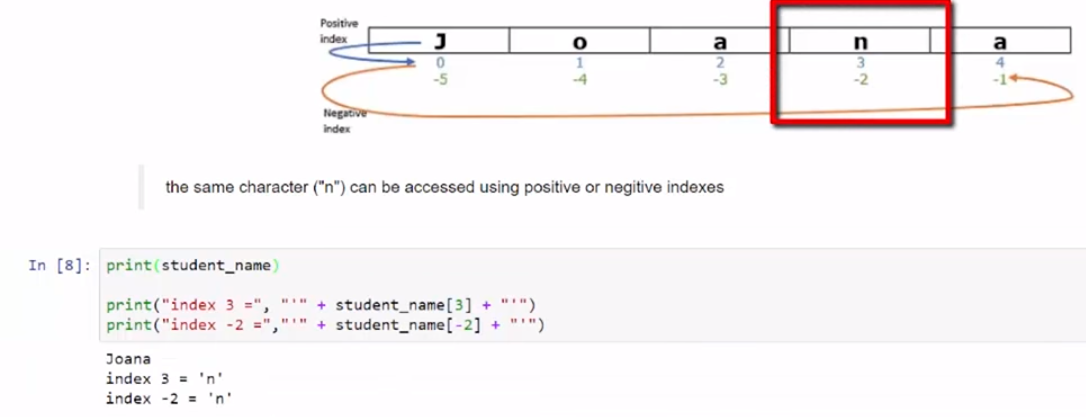
**Here we have a variable called end\_letter and we're just gonna assign that to the student\_name letter. So we can return a character with the indexing and assign it to a variable.** And so we expect that will return the same as the statement above. And then we can write that the student name ends with, and then we can put that end letter in there. So let's run that code. We see "Joanna" ends with "a", just as we expected.



Here is another example, where we say second to last letter is equal to negative two. So we have the first letters going In this direction, and then the last letter starting negative one, negative two, negative three. So second to last is negative two, so we can print that code. And we see that that second to last letter is the "n". And any of these characters can be accessed with positive or negative indexes.



Let's look at one last scenario, where we print the student name and then we can see the indexes at three (3) and minus two (-2), which is in the middle of our string. **And we see that both of those resolve to the "N" and proving that we can access the same character spot by index, either positive or negative.**



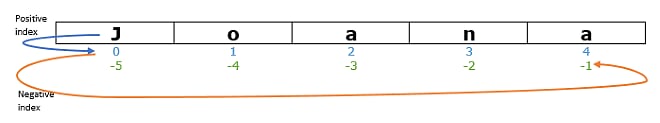
# Concept

## Using a negative index

### Access the end of a string using -1

Strings assign an **index** number address to each string character

* first character in a string is index 0
* last character in a string is index **-1**



To access the last character in a string

student\_name[-1]

## Examples

#### access the last character with the -1 index

negative index counts back from the last character in a string

# [ ] review and run example

student\_name = "Joana"

# get last letter

end\_letter = student\_name[-1]

print(student\_name,"ends with", "'" + end\_letter + "'")

# [ ] review and run example

# get second to last letter

second\_last\_letter = student\_name[-2]

print(student\_name,"has 2nd to last letter of", "'" + second\_last\_letter + "'")

# [ ] review and run example

# you can get to the same letter with index counting + or -

print("for", student\_name)

print("index 3 =", "'" + student\_name[3] + "'")

print("index -2 =","'" + student\_name[-2] + "'")

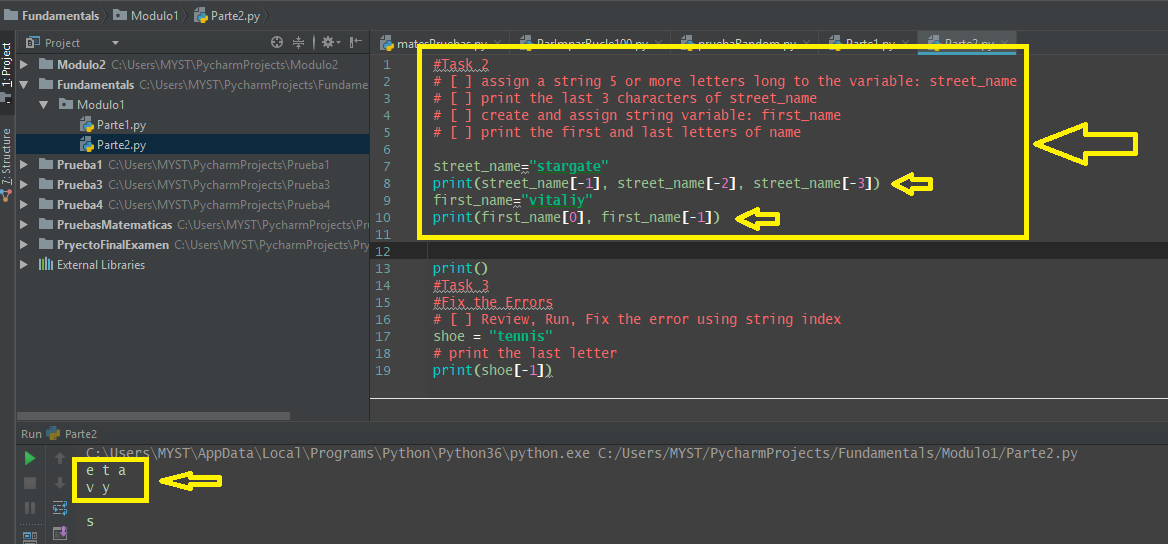
## Task 2

# [ ] assign a string 5 or more letters long to the variable: street\_name

# [ ] print the last 3 characters of street\_name

# [ ] create and assign string variable: first\_name

# [ ] print the first and last letters of name



## Task 3

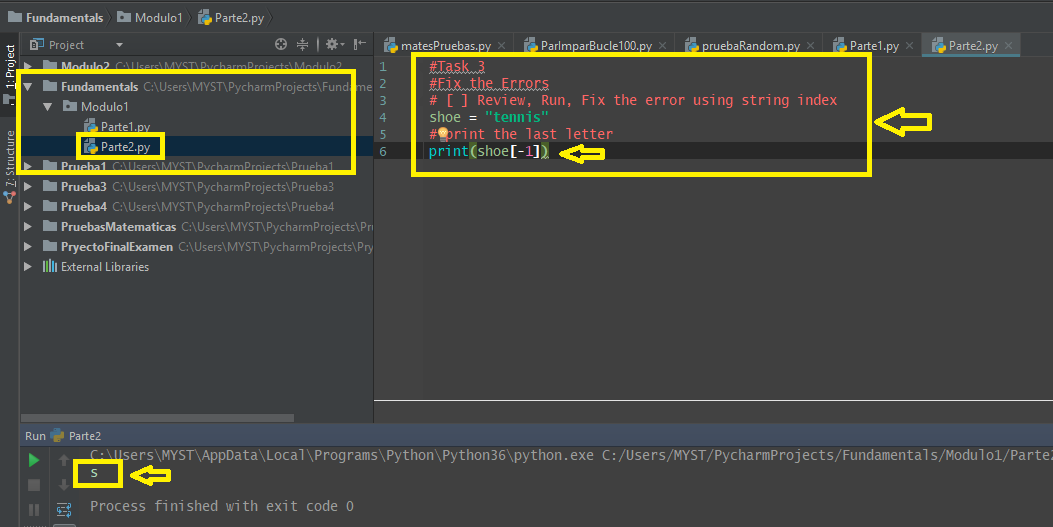
## Fix the Errors

# [ ] Review, Run, Fix the error using string index

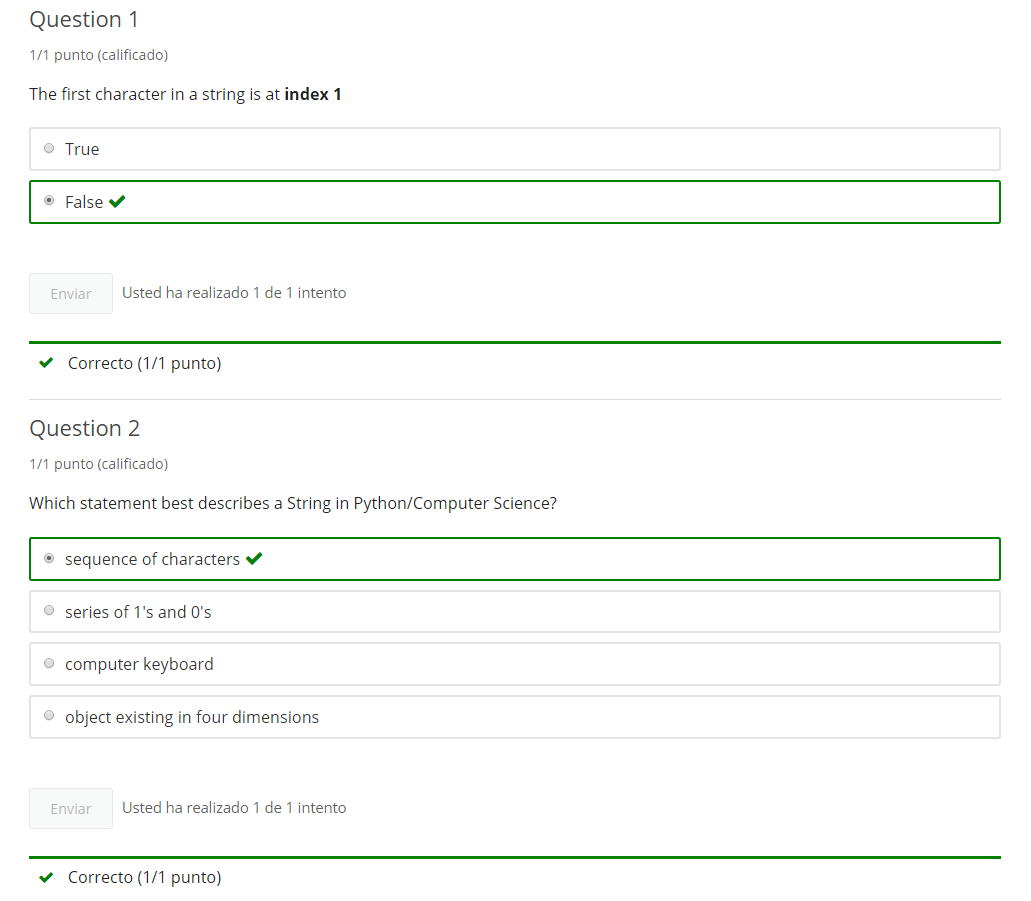
shoe = "tennis"

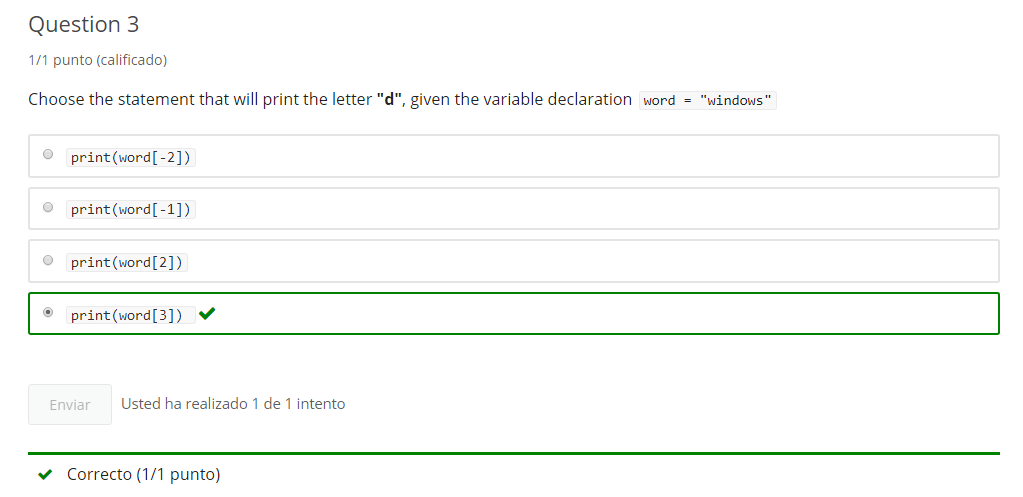
# print the last letter

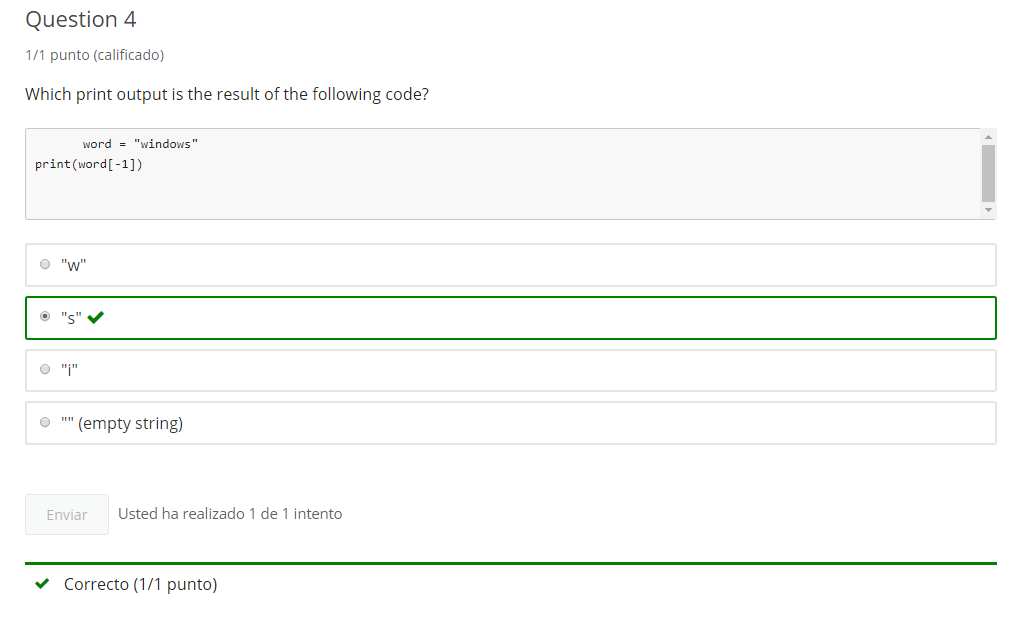
print(shoe(-1))



**2.4 Self-Check: Module 1: Lesson 1.1 (Graded)**







**3. Index Slicing**

**3.1 Intro Python**

Jupyter Notebook: Mod1\_2-1.2\_Intro\_Python.ipynb

The link to the .ipynb Jupyter Notebook files are in the last lesson of section 0 of module 1

## Sequence: String

* Accessing String Character with index
* **Accessing sub-strings with index slicing**
* Iterating through Characters of a String
* More String Methods

# Student will be able to

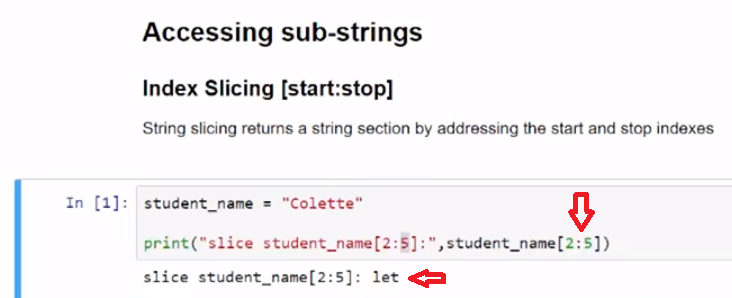
* **Slice strings into substrings**

**3.2 Accessing sub-strings**

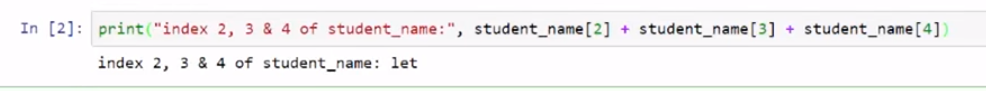
**Video:** **AccesingSubStringsV3.mp4**

**We can get sub-strings from within a string using index slicing, this allows us to get more than an individual character.** We have an example of an index slicing code.

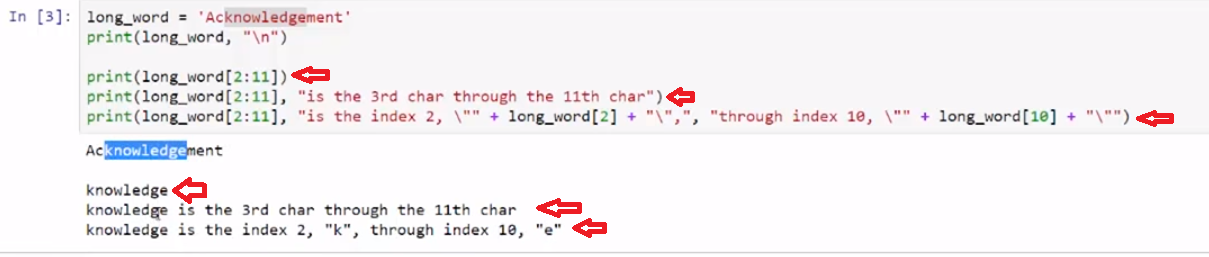
In this first cell, where we set the student name to "Colette", and we're going to access a slice from the index 2 for the start value of student\_name. And so we use the square brackets to do that. So this is my start value, so I'll start at index 2. So "C" would be index zero, so then "o", "l" and then, our stop value is index 5 and the index 5 is (counting) zero, one, tw,o three, four, five would go to that "t". **So we're gonna go from index 2 to the five, but we don't include the stop value, we stop at that value (5). So, this will give us the three letters, starting at index 2 and it ends at, and not includes index 5.** So, let's go ahead and run that code cell. We see that the two through five is indexes 2, 3 and 4, and stops at index 5.



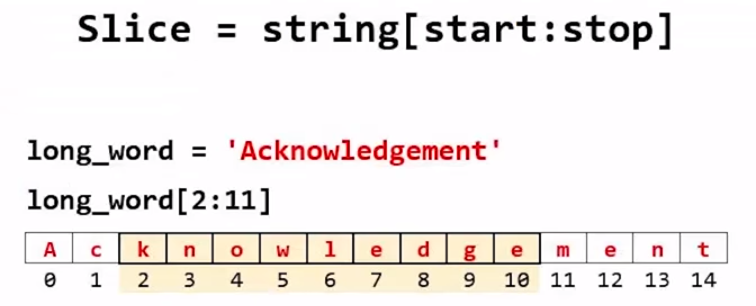
In this next example, we're just gonna print some different indexes there to show the point of 2, 3 and 4 were the same result of "let".



In this last example, we use a longer word calledlong\_word as our variable. And we're gonna go ahead and print long\_word and then we're gonna do a slice. So index two is the third character (coounting) zero, one, two and then we stop at the 11th character. Because we go to index 11 and that would be two through ten (2 - 10) lets go ahead and (counting) zero, one, two.. so it's start with the "K". And three, four, five, six,seven, eight, nine, ten, so we should see the word "knowledge" let's go ahead and check this. And so we see "acknowledgement", and "knowledge" is the third character through the 11th character. So index 10 is the 11th character, and knowledge is index 2 for "K" and index 10 is the ending "e".



**String slicing is done with a start and a stop index. The start index is the beginning of the string that we're going to pullout and it will stop just before the stop index location.**



# Concept

## Accessing sub-strings

### Index Slicing [start:stop]

String slicing returns a string section by addressing the start and stop indexes

# assign string to student\_name

student\_name = "Colette"

# addressing the 3rd, 4th and 5th characters

student\_name[2:5]

The slice starts at index 2 and ends at index 5 (but does not include index 5)

## Examples

# [ ] review and run example

# assign string to student\_name

student\_name = "Colette"

# addressing the 3rd, 4th and 5th characters using a slice

print("slice student\_name[2:5]:",student\_name[2:5])

# [ ] review and run example

# assign string to student\_name

student\_name = "Colette"

# addressing the 3rd, 4th and 5th characters individually

print("index 2, 3 & 4 of student\_name:", student\_name[2] + student\_name[3] + student\_name[4])

# [ ] review and run example

long\_word = 'Acknowledgement'

print(long\_word[2:11])

print(long\_word[2:11], "is the 3rd char through the 11th char")

print(long\_word[2:11], "is the index 2, \"" + long\_word[2] + "\",", "through index 10, \"" + long\_word[10] + "\"")

## Task 1

## slice a string

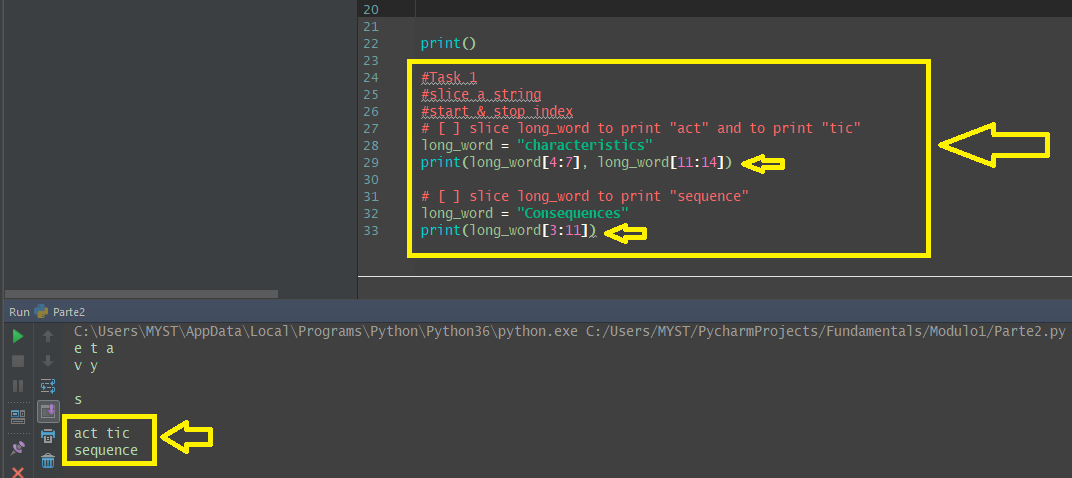
### start & stop index

# [ ] slice long\_word to print "act" and to print "tic"

long\_word = "characteristics"

# [ ] slice long\_word to print "sequence"

long\_word = "Consequences"



**3.3 Access start of sub-strings**

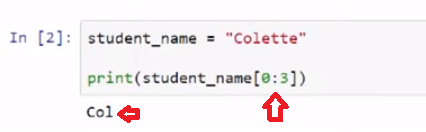
**Video:** **AcsessStartSubStringsV4.mp4**

**We can get just the beginning part of a string as a sub-string by using index slicing with only the stop index.**

**In our first example, we have the student\_name = "Colette". And we're gonna print a slice that has no start value indicated, and only shows a stop value of 3.** And this is the way we slice from the beginning of a string by default. **So if we put in a 0 value, it would be the same as leaving this first start value blank.** So let's go ahead and run that code. **And we see that we get (counting) 0, 1, and 2, and we stop at index 3, we do not include that "e". So we went from 0 andup to 2 and stopped at 3.**



**If we put a 0 in that location, we see we get the same result.**

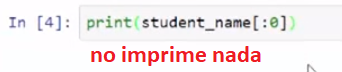


So the indexing will use this format of the square brackets with a colon and then the stop value.

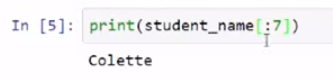
So let's look at printing just the first character. We get one character and that is character index 0.



**And if we print to index 0 as the stop value then we print nothing. So we end always at the final value.**



So with Colette, (counting) one, two,three, four, five, six, seven letters, if we put a 7 in there, we do not print the seventh index, because that is out of range. But we get "Colette". And so it stops at index 7, which is past the "e".



Default start for a slice is at index 0.

# Concept

## Accessing beginning of sub-strings

### Index Slicing [:stop]

String slicing returns a string section from index 0 by addressing only the stop index

student\_name = "Colette"

# addressing the 1st, 2nd & 3rd characters

student\_name[:3]

**default start for a slice is index 0**

## Example

# [ ] review and run example

student\_name = "Colette"

# addressing the 1st, 2nd & 3rd characters

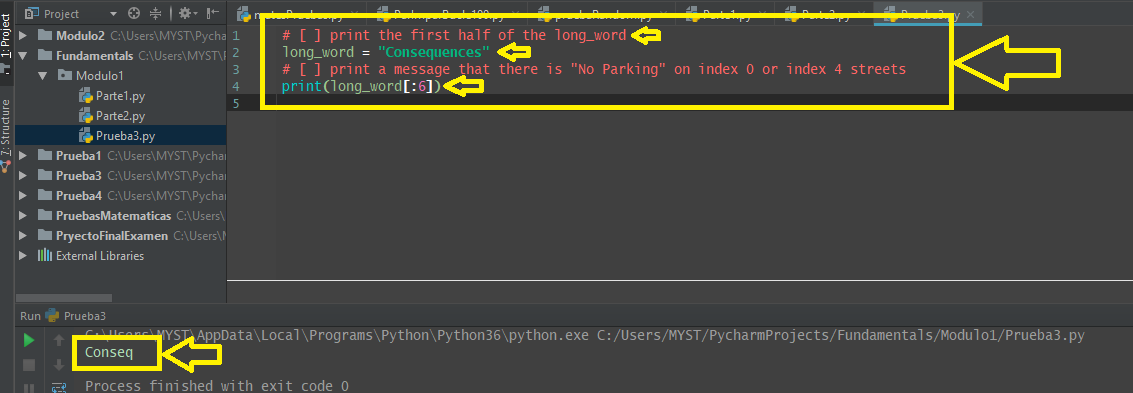
print(student\_name[:3])

## Task 2

# [ ] print the first half of the long\_word

long\_word = "Consequences"

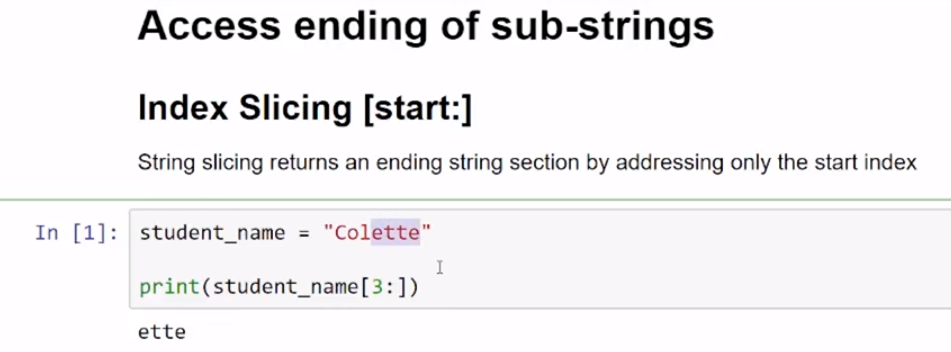
# [ ] print a message that there is "No Parking" on index 0 or index 4 streets



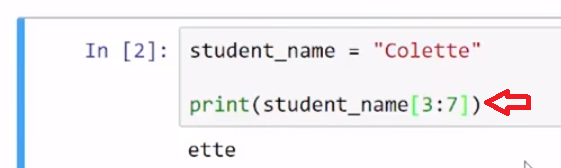
**3.4 Access end of sub-strings**

**Video:** **AcsessEndOfSubStringsV5.mp4**

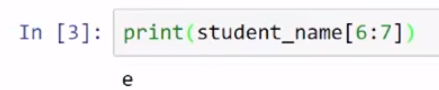
**The end part of a string could be returned by using a substring with only the start index. In this first cell, we have a student name "Colette", and we're going to take a string slice from three to no value, specified for the ending value.** We're gonna look at that and see that it goes through the end of the string, so whenever you use this empty value here, it will return all the way through the end of the strings.



It would be the same as if we went (counting) zero, one, two, three, four, five, six. **We're going to include six, so we're gonna put a stop value of seven, and we see we get the same result.**



So if we print the last character, which we know would be [6:7], we can see the "e" in "Colette" returned,



but we can also get that result just by entering no value.



So a blank ending index is gonna return all the way through the last character

# Concept

## Accessing ending of sub-strings

### Index Slicing [start:]

String slicing returns a string section including by addressing only the start index

student\_name = "Colette"

# addressing the 4th, 5th and 6th characters

student\_name[3:]

**default end index returns up to and including the last string character**

## Example

# [ ] review and run example

student\_name = "Colette"

# 4th, 5th, 6th and 7th characters

student\_name[3:]

## Task 3

# [ ] print the second half of the long\_word

long\_word = "Consequences"

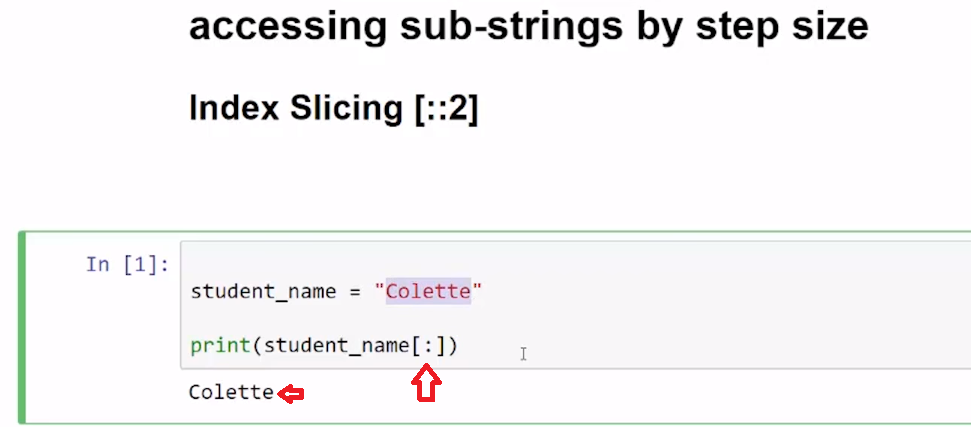
# [ ] print a message that there is "No Parking" on index 0 or index 4 streets

**3.5 Access sub-strings by Step size**

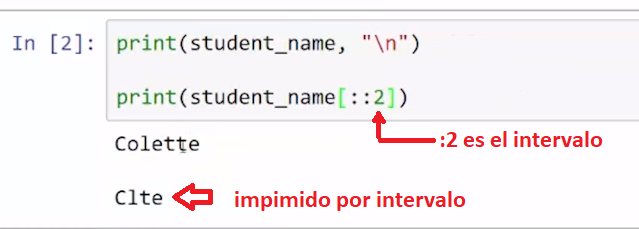
**Video:** **AccessSubStringsStepSizeV6.mp4**

**If we wanna get every other character in a string or some interval of characters we can use the step size in string slicing.**

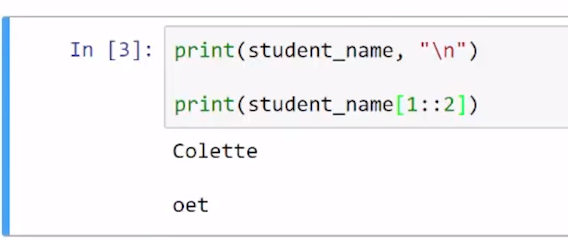
**So what happens when we have a string and we tried to slice it with no values included? Well, we get the entire string returned**, with the default start being the first character, or index zero, and the default end going past the last character, so that we return the entire string.



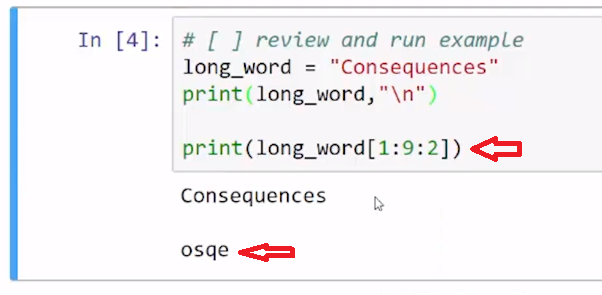
In this example, **we're gonna add a third element and that's called the step. So we have our start value, our stop value and then this 2 is a step.** **And that tells you how many we will skip as the interval, as we're counting from the start value to the stop value.** So lets go ahead and look at what the step of 2 is with the default to 0 to the end of the string. **And we see that at the 0 index and then we go to the 2 index, the 4 index and the 6 index, because we step by 2.**



**In this example we start with the second character, which is at index we start 0, 1 and then we're gonna take a step of 2**. So let's look at that one,so we started at index 1, and then we went to 3 and 5,and then we are ended and so we just get the oet return.



This final one with long\_word using "consequences", we're going to start at index 1, which is the second character. And then go through index 9 or the ninth character rather and stopping at index 9. So (counting) zero, one, two, three, four, five, six, seven, eight, and then we'll stop at nine. And then we're gonna get every other character, and let's run that one. And we get letter "o" at index 1, and then "s", and then "q", and then "e", and then we stop. So we can still use the start and end values.



# Concept

## accessing sub-strings by step size

### Index Slicing [:], [::2]

* **[:]** returns the entire string
* **[::2]** returns the first char and then steps to every other char in the string
* **[1::3]** returns the second char and then steps to every third char in the string

the number **2**, in the print statement below, represents the **step**

print(long\_word[::2])

## Examples

# [ ] review and run example

student\_name = "Colette"

# return all

print(student\_name[:])

# [ ] review and run example

student\_name = "Colette"

# return every other

print(student\_name[::2])

# [ ] review and run example

student\_name = "Colette"

# return every third, starting at 2nd character

print(student\_name[1::2])

# [ ] review and run example

long\_word = "Consequences"

# starting at 2nd char (index 1) to 9th character, return every other character

print(long\_word[1:9:2])

## Task 4

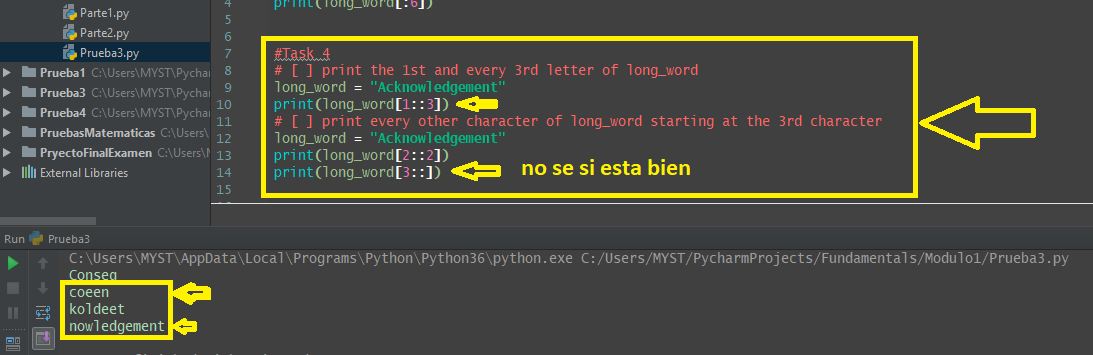
# [ ] print the 1st and every 3rd letter of long\_word

long\_word = "Acknowledgement"

# [ ] print every other character of long\_word starting at the 3rd character

long\_word = "Acknowledgement"

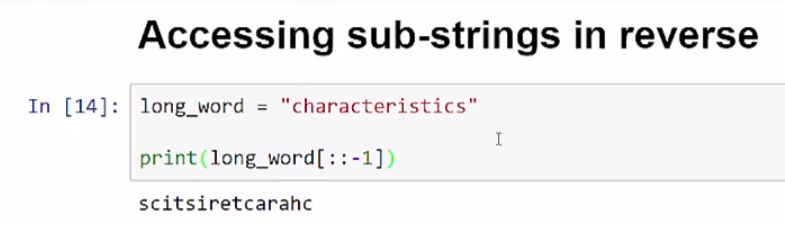
print(long\_word[2::2])



**3.6 Access sub-string step backwards**

**Video:** **AcessSubStringsStepBackwardsV7.mp4**

**We can easily reverse a string by using a step size of negative one (-1).** So in our first example using a negative step value, we're going to look at long\_word, a variable that has 15-letter string characteristics, and **we use the negative step.** So let's look at the start value and the end value and see what these look like. So, the default here**, it seems to start with the last character and end with first character.**



**Let's look at our second example here where we start at the sixth index**, so zero, one, two, three, four, five, six, is the "t". **And taking a negative step again, and we see that it started at the "t", and went back to the print the first character. And so, it just stops before that first one.**



**So, if we want to easily reverse a string, or we want to iterate backwards through each character in a string, we can use the negative one step value.**

# Concept

## Accessing sub-strings continued

### stepping backwards

print(long\_word[::-1])

use **[::-1]** to reverse a string

## Example

# [ ] review and run example of stepping backwards using [::-1]

long\_word = "characteristics"

# make the step increment -1 to step backwards

print(long\_word[::-1])

# [ ] review and run example of stepping backwards using [6::-1]

long\_word = "characteristics"

# start at the 7th letter backwards to start

print(long\_word[6::-1])

## Task 5

use slicing

# [ ] reverse long\_word

long\_word = "stressed"

# [ ] print the first 5 letters of long\_word in reverse

long\_word = "characteristics"

## Task 6

use slicing

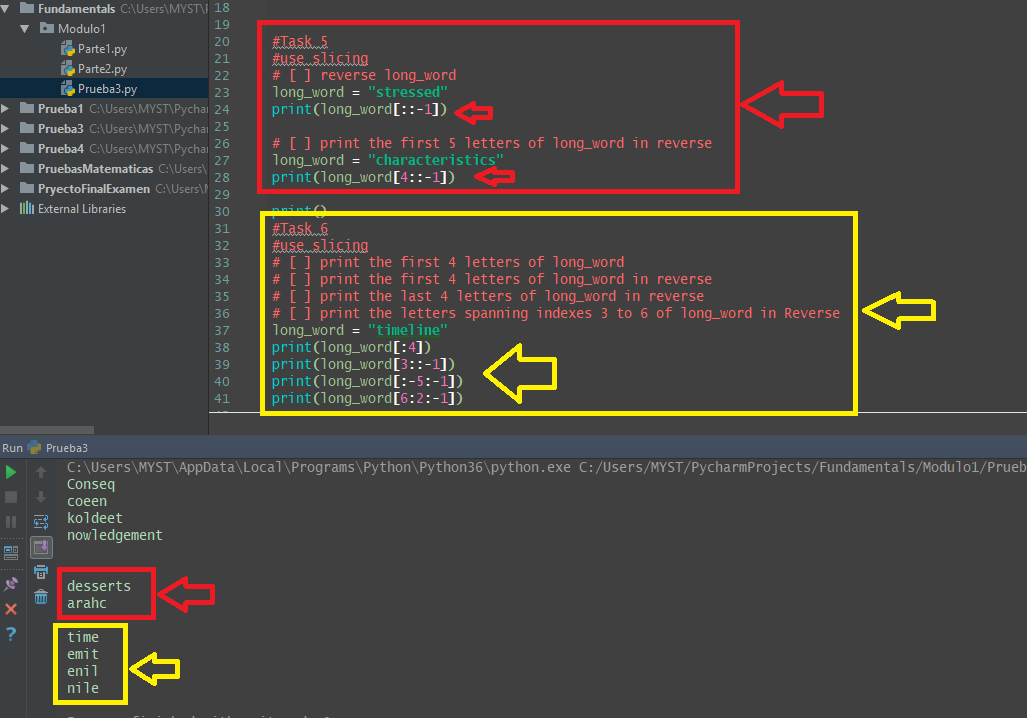
# [ ] print the first 4 letters of long\_word

# [ ] print the first 4 letters of long\_word in reverse

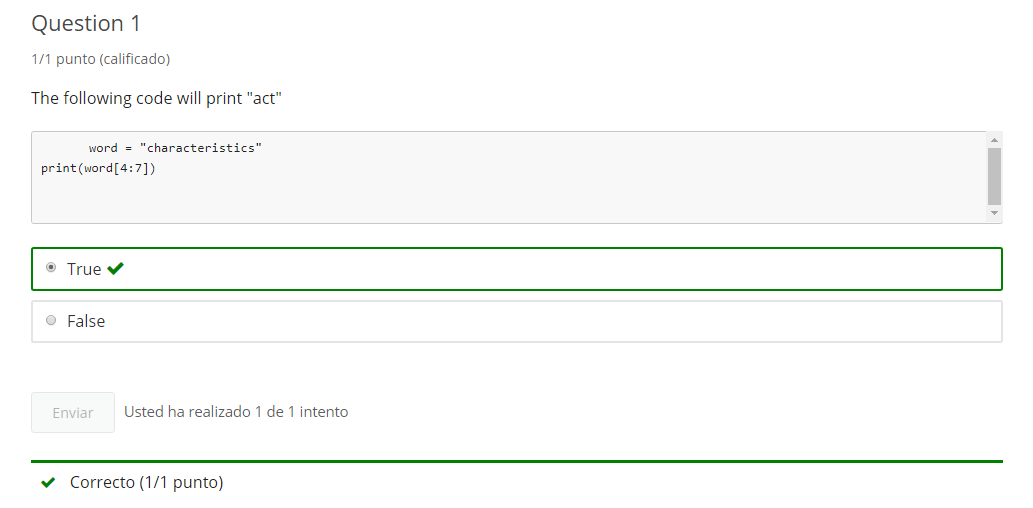
# [ ] print the last 4 letters of long\_word in reverse

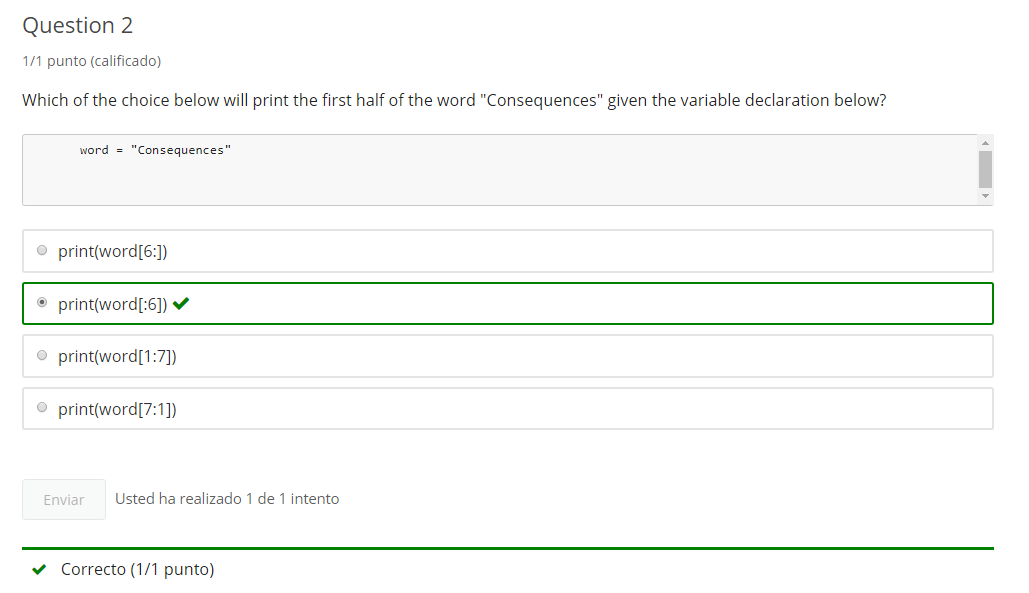
# [ ] print the letters spanning indexes 3 to 6 of long\_word in Reverse

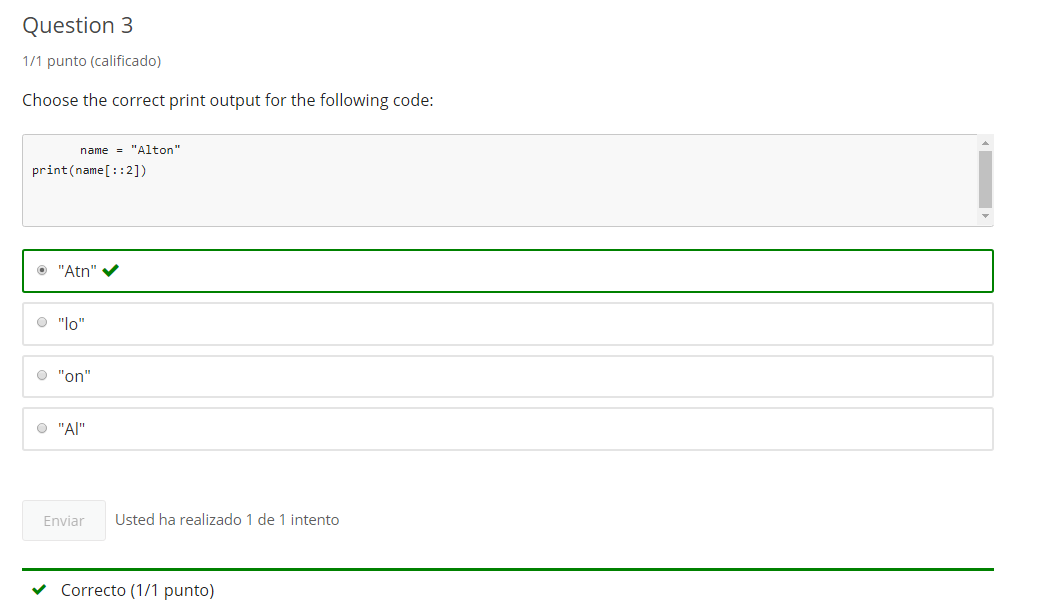
long\_word = "timeline"

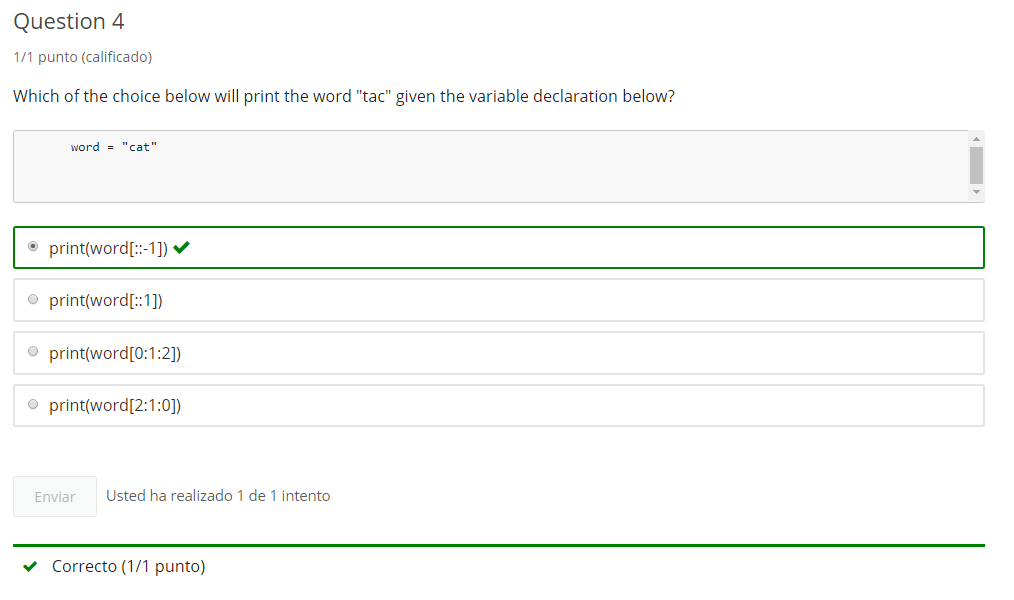


**3.7 Self-Check: Module 1: Lesson 1.2**









**4. Iterating Strings**

**4.1 Python Intro**

Jupyter Notebook: Mod1\_2-1.3\_Intro\_Python.ipynb

The link to the .ipynb Jupyter Notebook files are in the last lesson of section 0 of module 1

## Sequence: String

* Accessing String Character with index
* Accessing sub-strings with index slicing
* **Iterating through Characters of a String**
* More String Methods

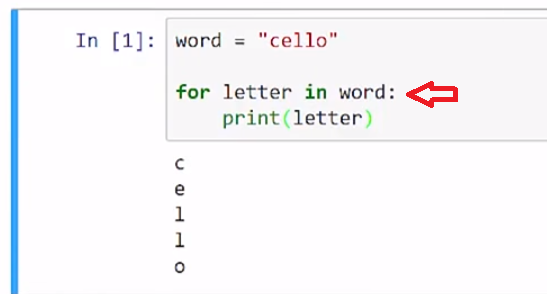
# Student will be able to

* **Iterate through String Character**

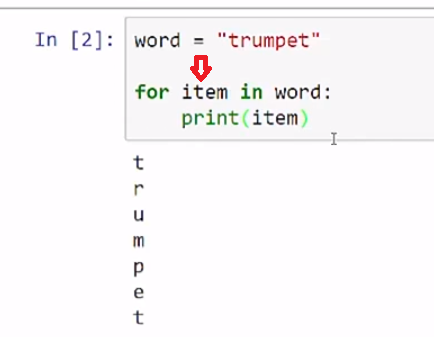
**4.2 Iterate a String: 1 character at a time**

**Video:** **IterateString1ChacarcterAtTimeV8.mp4**

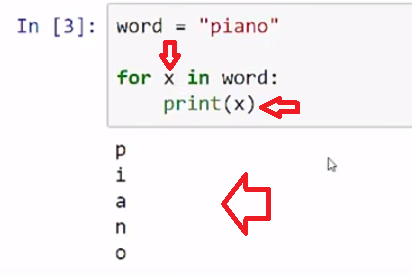
**To iterate through a string one character at a time, we can use the for/in statement.** As we look at our first code cell that has a for loop using the "in" keyword. We are gonna get a variable that we assign a string. In this case, it is the word "cello", and **what happens in this for loop is it's gonna iterate through this string that we've provided the word "cello" and iterate through each character in that word.** So it's gonna, first, get the firstcharacter, a c and then a e, then l. And the first time through, when it's gonna do with that "e" that it grabs, it's gonna assign it to this variable letter. And this arbitrary name that we've put there. We can put any validvariable name there. And so, then what happens it's gonna print the "c". And I'll go back through and get the next character the "e", and that will be a sign to letter, and so we can see that run right here, and so print out "cello" each letter at a time.



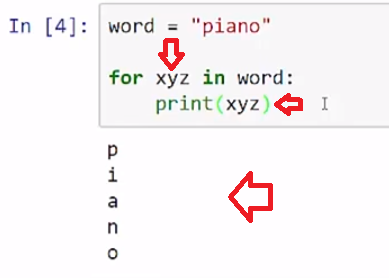
**And I can do the same type of loop without using letter because, again, it's not a keyword, it's justa variable name that I defined.** And so, I have to keep the "word" variable because word contains my string. And that's what I'm gonna iterate through. **But for each character that's gonna be returned as I iterate through, I can put whatever valid variable name I like. So now I'm changing that to "item".** And we see that the same code can run using "item".



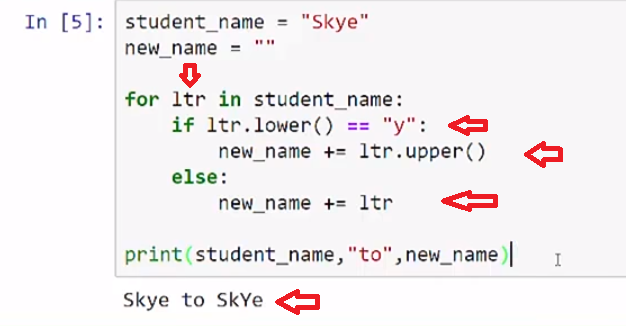
And just one final time, here I have a word, again, it could be still "cello" or "trumpet" but I changed it to "piano". And, I just put a "x" in here for the iteration variable, and so let's go ahead and run that. And see that works as piano



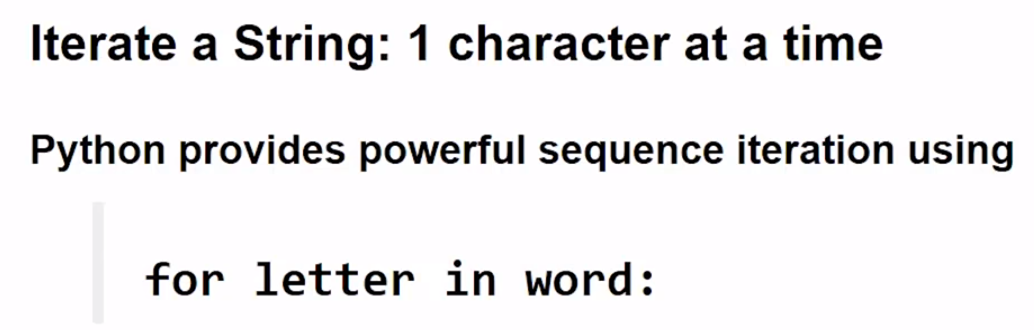
and I could call it "xyz", and it will still works because it really doesn't matter what I use as long as I use valid variable name, so you see that, we'll run again.



In this final example here, I used a student\_name variable and I have the name "Skye", and then I have a new\_name variable that is empty string. What happens is we're gonna iterate through each of the characters in the name "Skye" and we're gonna assign it to that "ltr", standing for letter, temporary variable, so it comes through. And so it gets the "S" that goes through, and so it says, okay, here's the "S". Let me put it to lower case and compare it to this lower case "y", if it is equal then we'll run this code but it's not equal, so what we're gonna do is just add that letter to this empty string. So its starts off with "s" and then it will add a "k", and when it gets to this "y" we'll execute this code right here which says, take that letter and make it upper case, so the end of that loop we print out the variable, the new\_name. So let's go ahead and look at that code. **So you see it can run through and change "Skye" to "SkYe" with a upper case "Y".**



And so, we can run a string test on those letters as we iterate through each character, we can run the string comparisons, and we can use a for loop with the "in" keyword to iterate through each character in the string.



# Concept

## Iterate a String: 1 character at a time

### for letter in word:

Python provides powerful sequence iteration features. Below, **for letter in word:** loops through each letter in word.

word = "cello"

for letter in word:

print(letter)

The variable **letter** is an arbitrary variable name . Any valid variable name can be used.

## Examples

# [ ] review and run example

word = "cello"

for letter in word:

print(letter)

# [ ] review and run example

# note: the variable 'letter' changed to 'item'

word = "trumpet"

for item in word:

print(item)

# [ ] review and run example

# note: variable is now 'xyz'

# using 'xyz', 'item' or 'letter' are all the same result

word = "piano"

for xyz in word:

print(xyz)

# [ ] review and run example

# creates a new string (new\_name) adding a letter (ltr) each loop

# Q?: how many times will the loop run?

student\_name = "Skye"

new\_name = ""

for ltr in student\_name:

if ltr.lower() == "y":

new\_name += ltr.upper()

else:

new\_name += ltr

print(student\_name,"to",new\_name)

## Task 1

## iterate a String

### one character at a time

# [ ] Get user input for first\_name

# [ ] iterate through letters in first\_name

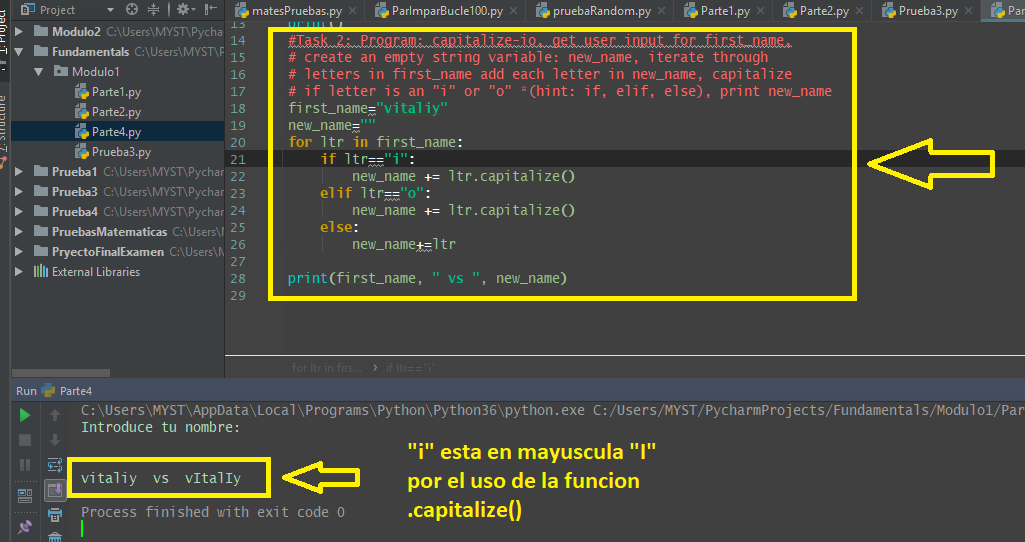
# - print each letter on a new line

## 

## Task 2

## Program: capitalize-io

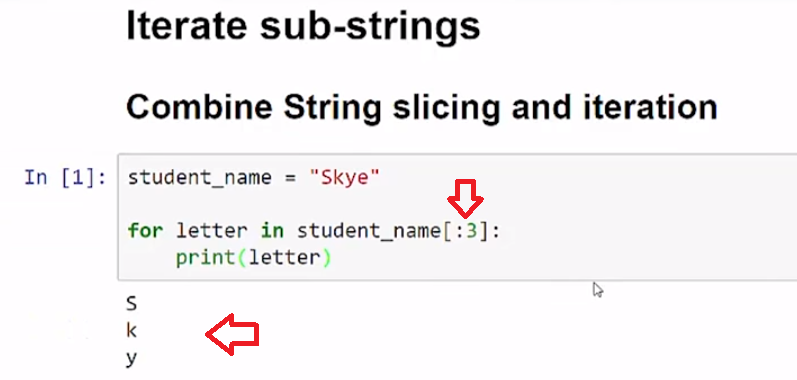
* get user input for first\_name
* create an empty string variable: new\_name
* iterate through letters in first\_name
  + add each letter in new\_name
  + capitalize if letter is an "i" or "o" \*(hint: if, elif, else)
* print new\_name



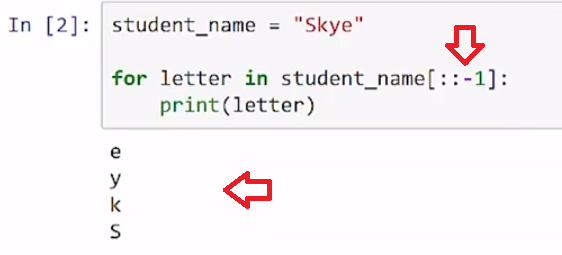
**4.3 Iterate by sub-string**

**Video:** **IterateBySubStingV9.mp4**

**We can Iterate through sub-strings by using a combination of string slicing and for/in loop.** In the first code cell we have a student\_name equal to the string "Skye", S-K-Y-E, the name. **And we're going to combine using the for/in Iteration with some string slicing so here use the student\_name we slice it. It's the default so it starts at index zero and it will stop at index 3.** So it will go to, up to the third character and include index two, so that should spell "Skye". Let's run that. So we see how that works.



**And then in the second code cell, we have here the same student name, but I'm using a slice that has a negative step.** **So that's gonna print my name in reverse.** So we can see that it's spelled at e, y, k, S.



And so we can use combinations of slicing with our for/in Iteration.

# Concept

## Iterate sub-strings

### Combine String slicing and iteration

student\_name = "Skye"

for letter in student\_name[:3]:

print(letter)

Iterate backwards using: **student\_name[::-1]**

## Example

# [ ] review and run example

student\_name = "Skye"

for letter in student\_name[:3]:

print(letter)

# Iterate BACKWARDS

# [ ] review and run example

student\_name = "Skye"

# start at "y" (student\_name[2]), iterate backwards

for letter in student\_name[2::-1]:

print(letter)

## Task 3

### String slicing and iteration

# [ ] create & print a variable, other\_word, made of every other letter in long\_word

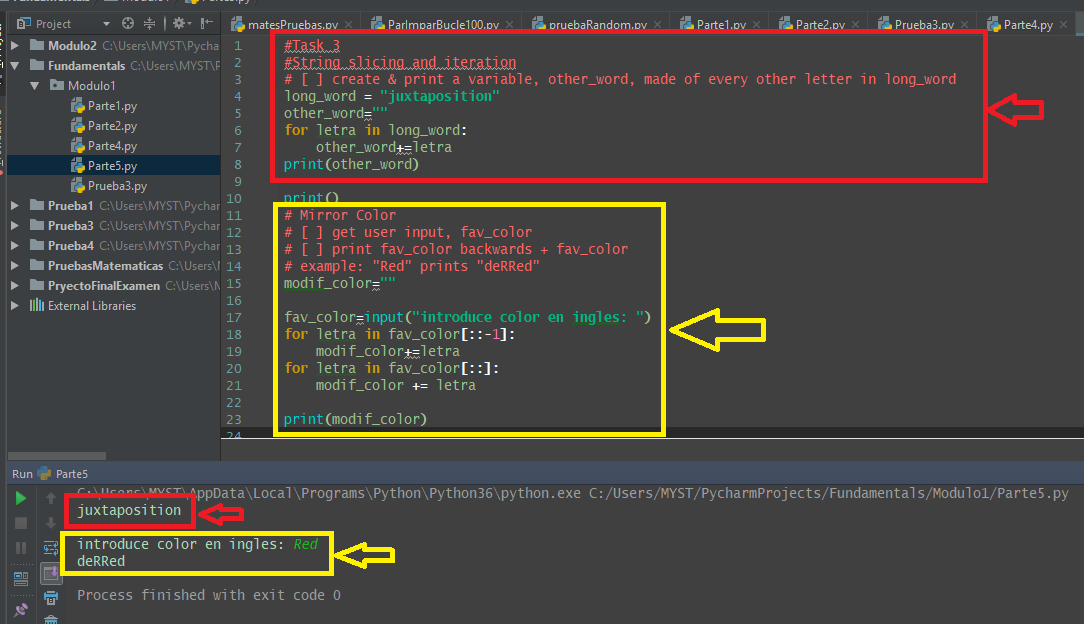
long\_word = "juxtaposition"

# Mirror Color

# [ ] get user input, fav\_color

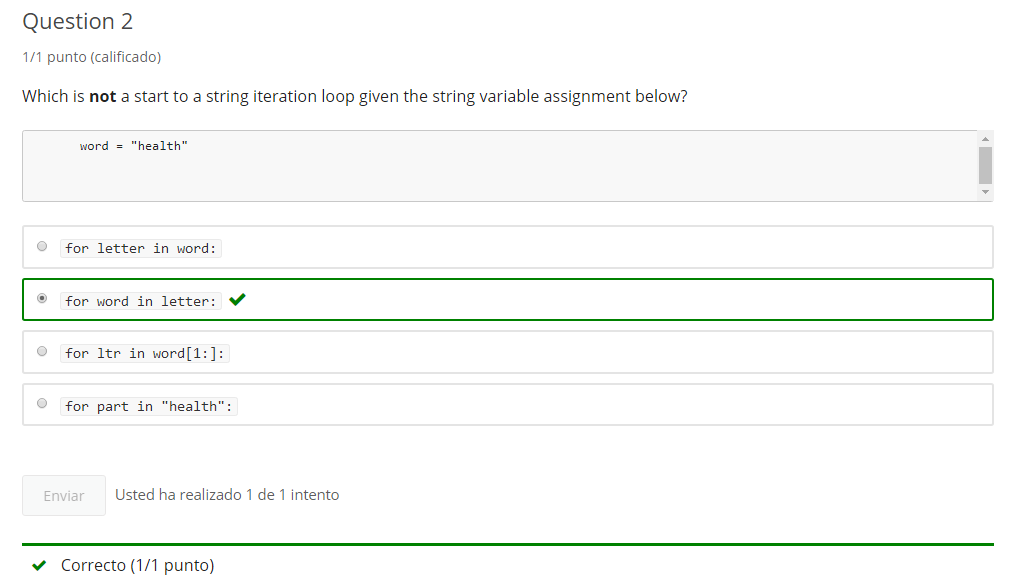
# [ ] print fav\_color backwards + fav\_color

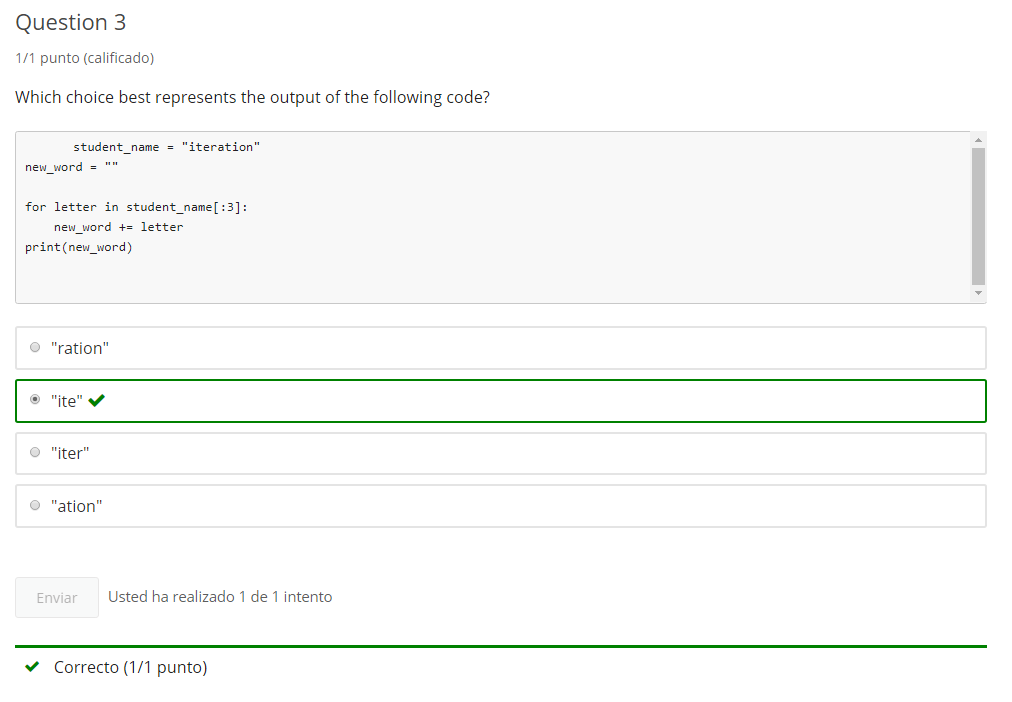
# example: "Red" prints "deRRed"

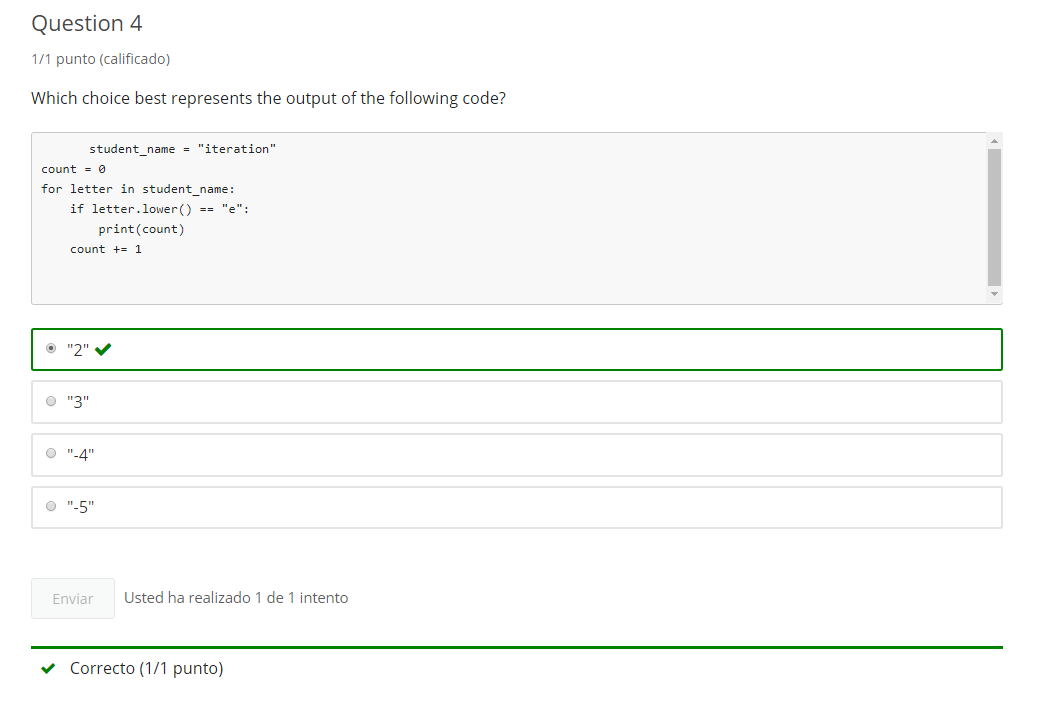


**4.4 Self-Check: Module 1: Lesson 1.3**









**5. Strings Methods**

**5.1 Intro Python**

Jupyter Notebook: Mod1\_2-1.4\_Intro\_Python.ipynb

The link to the .ipynb Jupyter Notebook files are in the last lesson of section 0 of module 1

## Sequence: String

* Accessing String Character with index
* Accessing sub-strings with index slicing
* Iterating through Characters of a String
* **More String Methods**

# Student will be able to

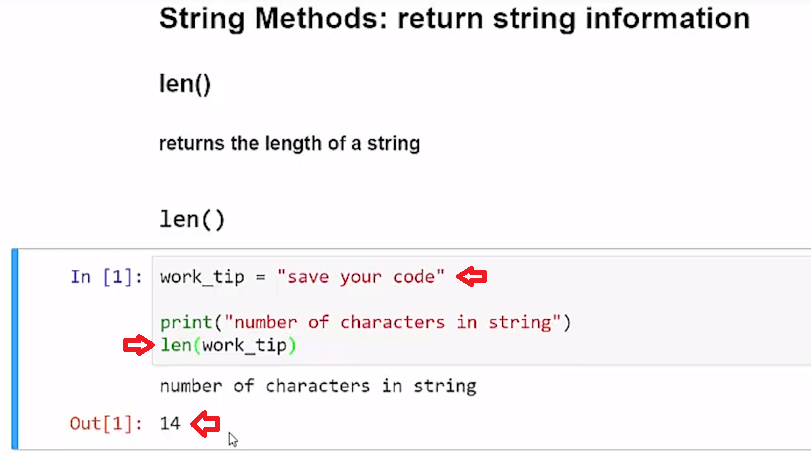
* **Use String  Methods**
* len()
* .count()
* .find()

**5.2 Return String information**

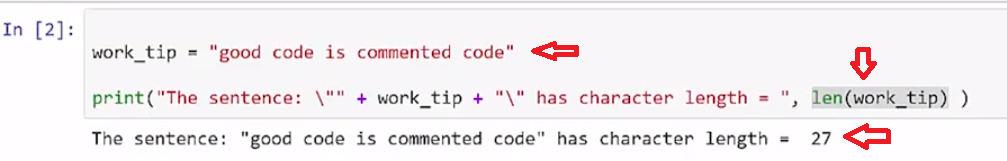
**String Methods: Length()**

**Video:** **StringMethodLengthV10.mp4**

**To return the length of a string, we'll use the len() function.** In this first code cell, we have a variable, work\_tip. With the string, "save your code", which is a good habit. I'll save, and **we're gonna find the length of that string using the length function, len, followed by parenthesis and a string. You can put a literal string or the variable as I placed in here.** We run that, we see save your code including the spaces and any other characters, formatting would be 14 characters.

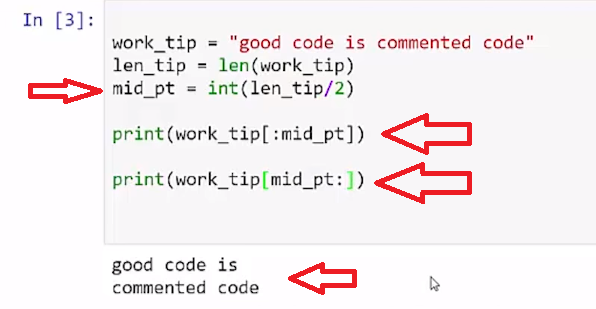


**Now we'd like to use the work\_tip length in a print function, and I have a new work\_tip which is "good code, is commented code". We can find the length of that printed out in a print statement because this returns an integer value**, and so the integer can be displayed ina print statement with commas. So, let's run that code, and we see the work\_tip is printed as well as the integer value, which is 27 characters.



We're gonna use that in the next example. **This 27 character work\_tip code, that's a little too long for me, so I'd like to cut that in half**, and so I'll find that length again, which this should **return the 27, and then I'm gonna divide that by 2. Dividing an integer by 2 will result in a float value, even if it has no decimal place, values, it will still put the decimal place .00 and change that to a float value. For that reason I'm casting it back to an int(), so that I can use the integer to do some string slicing below.** Let's run this, and we see **that I Get the first half of the string in the first print statement. I do that with the string slice. I start with the default value blank. That means start at index zero or the first character and stop right print through the Character up to the stop value so that would be that halfway point. The integer would be created from this flow value dividing by two and rounding that.**

**The second print statement will start at that index point, of the midpoint, and go to the end of the code** and we see that run here.



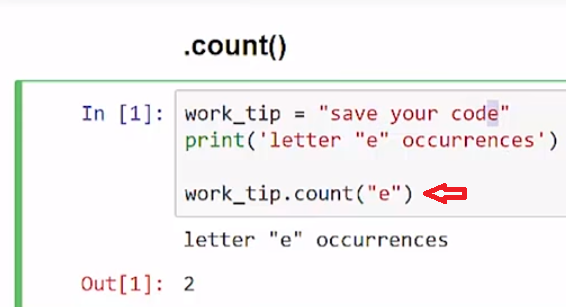
We've seen that the length function len() can return the length of a string as an integer. We can use that in calculations and in print statements.

**String Methods: .Count()**

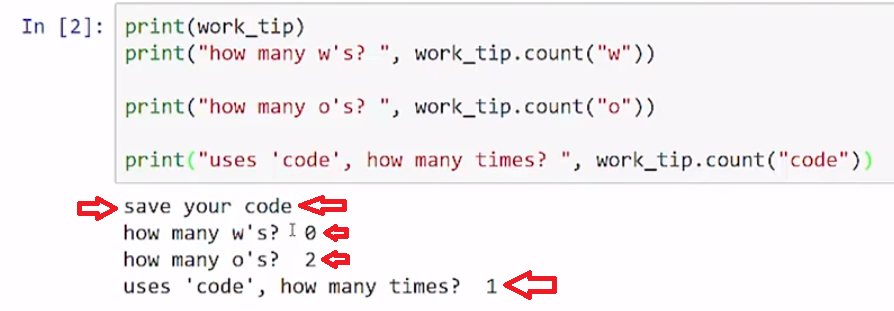
**Video:** **StringMethodCountV11.mp4**

**To determine the number of times that a character or a sub-string occurs within a string, we use the count method.**

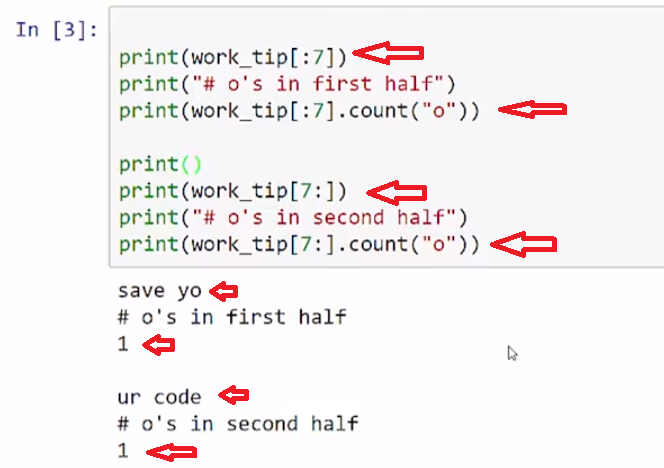
In our first code cell, we're gonna look at the string work\_tip Save your code. **And we're going to use the count method, which is a string method, and we'll look within the string work\_tip, for the occurrence of the letter "e".** Let's run the code. So we see the letter "e" occurs two times on that work\_tip, so that would be the "save code" "e" is there.



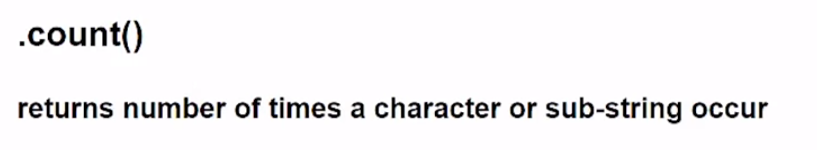
**We can also look for other letters and we can see that some letters will not occur, and we can search for strings of multiple characters.** So in this case, we're gonna look for the string C-O-D-E for "code" and see how that occurs. And so we see printed our work\_tips, "save your code", that there are no "w", there are two "o", and that the string "code" occurs in this work\_tip string one time.



**Save your code is 14 characters long, and so we're gonna break this in two halves. So using string slicing, we can start at the index zero, the first character and go to index six, because we'll stop at index seven. And then we'll also do the second half starting at index seven going to the end of the string.** You'll see that output is looking for letter "o", there is one o in the first half so it stops at "save" then "yo", and then the rest of "your" is over here, "ur", and there's an "o" there.



So we are able to combine both string slicing and our count method, because the string slice returns a string and we can use the count method on any string.



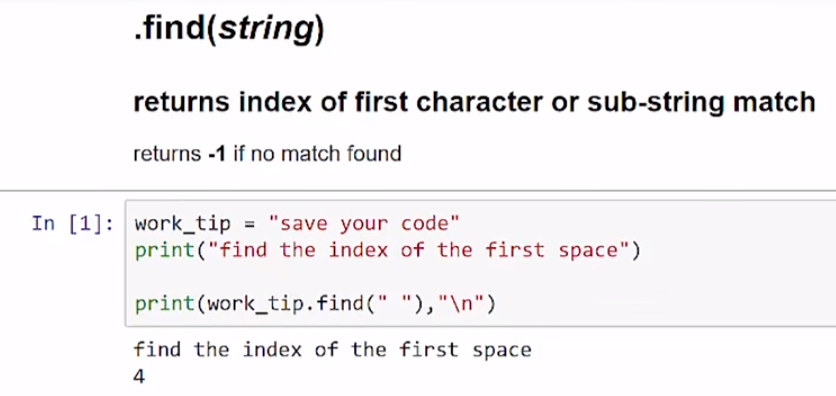
Count returns the number of times a character or a sub-string occur within a string.

**String Methods: .Find()**

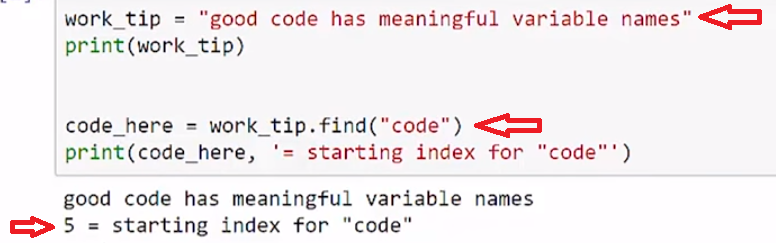
**Video:** **StringMethodFindV12.mp4**

**To find the index of where a character or a sub-string occurs within a string, we can use the find method.**

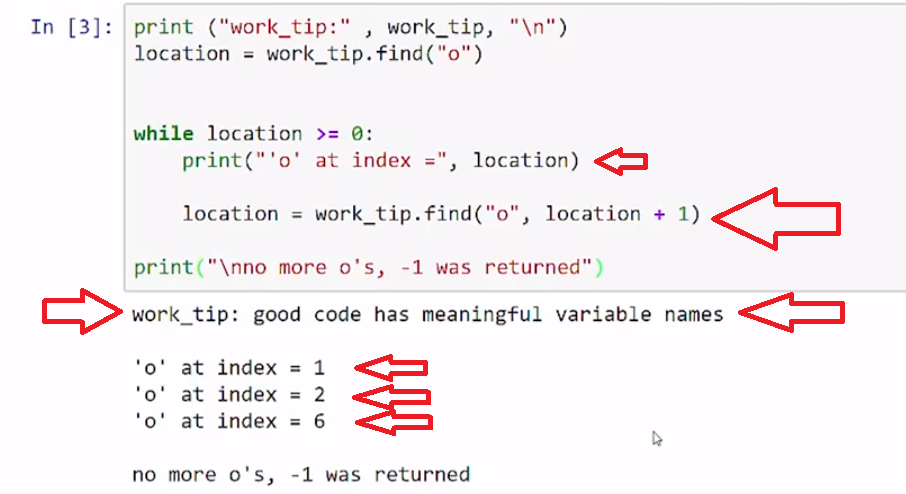
In our first code cell, we use the find method on work\_tip which is a variable that contains the string "save your code". Don't forget, save your code. **And we see that it is searching for a space and we have a space right there at index four.** So this is "s" is (counting) zero, one, two, three**. So the space is index four.** Let's run the code, and we see the first space is the index four. And so, the find method is very useful because you can move within a string to a index because now I have that index location.



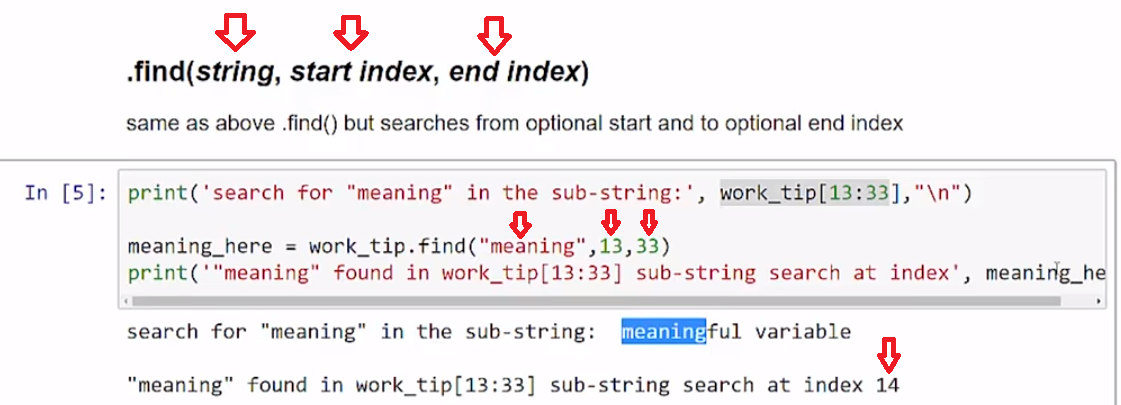
Here we have another work\_tip which is "good code has meaningful variable names." And this work\_tip here tells me what that variable is, it's a string work\_tip. I am gonna print it out but then **I wanna find the location of the word "code" and what we'll do is tell me where the first character of that match is at, and so, we see there with the "c" in "code".** And so, that's one past our white space that we had, that was at four, so we see now that's at five.



Let's run the code and then walk through it a little bit. **Before we enter the loop, we see if there are any "o"s. So this is going to find an "o" at the index 1, the second character.** **And so this is gonna come in as a 1, and so it says, "is 1 greater than 0?" so it enters the loop. Then it's gonna find the second index location. But the only way it can find the second index location is by starting the search from the location that it initially found plus one (+ 1), so it's gonna move to start searching from index 2 on.** That's what this location "+ 1" means, its location here was 1, now it's 2. **And so the next time through the loop, it's gonna go "2 + 1", and so that's gonna be from the third index on. So it's gonna go until it finds the next "code". And so that's how we get to the sixth and to the end, when it gets no more "o"'s to be found**, returns a -1 and exits the loop.



**So here we're gonna look at example where the find method's going to take three arguments. So the first argument is this string, and so we're gonna search for "meaning", and then we're going to go to a start index and then an end index, so this is, basically, it's searching within the slice between 13 and the ending of 33, so it's very much gonna look through this slice like this but it's not going to use the slice, it's going to use this format of end or beginning and end, as arguments for the find method.** So let's go ahead andrun that code, and we see that the meaningful variable that is this sub-string that we're gonna search through, and then it's gonna **search for "meaning"**, **and "meaning" is located at the first index there.** And so it says "meaning" was found in this sub-string at index, and it's the meaning\_here variable, it found meaning\_here at **index 14**.



The find method can be used to find a string or character in another string using a start index and an end index.

# Concept

## String Methods: return string information

### len()

returns a strings length

### .count()

returns number of times a character or sub-string occur

### .find()

returns index of first character or sub-string match  
returns **-1** if no match found

work\_tip = "save your code"

# number of characters

len(work\_tip)

# letter "e" occurrences

work\_tip.count("e")

# find the index of the first space

work\_tip.find(" ")

# find the index of "u" searching a slice work\_tip[3:6]

work\_tip.find("u",3,6)

These methods **return** information that we can use to sort or manipulate strings

## Examples

run each example cell in order

# [ ] review and run example

work\_tip = "save your code"

print("number of characters in string")

print(len(work\_tip),"\n")

print('letter "e" occurrences')

print(work\_tip.count("e"),"\n")

print("find the index of the first space")

print(work\_tip.find(" "),"\n")

print('find the index of "u" searching a slice work\_tip[3:9] -', work\_tip[3:9])

print(work\_tip.find("u",3,9),"\n")

print('find the index of "e" searching a slice work\_tip[4:] -', work\_tip[4:])

print(work\_tip.find("e",4))

### len()

returns a strings length

# [ ] review and run example

work\_tip = "good code is commented code"

print("The sentence: \"" + work\_tip + "\" has character length = ", len(work\_tip) )

# [ ] review and run example

# find the middle index

work\_tip = "good code is commented code"

mid\_pt = int(len(work\_tip)/2)

# print 1st half of sentence

print(work\_tip[:mid\_pt])

# print the 2nd half of sentence

print(work\_tip[mid\_pt:])

### .count()

returns number of times a character or sub-string occur

# [ ] review and run example

print(work\_tip)

print("how many w's? ", work\_tip.count("w"))

print("how many o's? ", work\_tip.count("o"))

print("uses 'code', how many times? ", work\_tip.count("code"))

# [ ] review and run example

print(work\_tip[:mid\_pt])

print("# o's in first half")

print(work\_tip[:mid\_pt].count("o"))

print()

print(work\_tip[mid\_pt:])

print("# o's in second half")

print(work\_tip[mid\_pt:].count("o"))

### .find(string)

returns index of first character or sub-string match  
returns **-1** if no match found

#### .find(string, start index, end index)

same as above .find() but searches from optional start and to optional end index

# [ ] review and run example

work\_tip = "good code has meaningful variable names"

print(work\_tip)

# index where first instance of "code" starts

code\_here = work\_tip.find("code")

print(code\_here, '= starting index for "code"')

# [ ] review and run example

# set start index = 13 and end index = 33

print('search for "meaning" in the sub-string:', work\_tip[13:33],"\n")

meaning\_here = work\_tip.find("meaning",13,33)

print('"meaning" found in work\_tip[13:33] sub-string search at index', meaning\_here)

# [ ] review and run example

# if .find("o") has No Match, -1 is returned

print ("work\_tip:" , work\_tip)

location = work\_tip.find("o")

# keeps looping until location = -1 (no "o" found)

while location >= 0:

print("'o' at index =", location)

# find("o", location + 1) looks for a "o" after index the first "o" was found

location = work\_tip.find("o", location + 1)

print("no more o's")

## Task 1

## len()

# [ ] use len() to find the midpoint of the string

# [ ] print the halves on separate lines

random\_tip = "wear a hat when it rains"

## Task 2

## .count()

# for letters: "e" and "a" in random\_tip

# [ ] print letter counts

# [ ] BONUS: print which letter is most frequent

random\_tip = "wear a hat when it rains"

## Task 3

### .find()

# [ ] print long\_word from the location of the first and second "t"

long\_word = "juxtaposition"

## Task 4

## Program: print each word in a quote

start = 0

space\_index = quote.find(" ")

while space\_index != -1:

# code to print word (index slice start:space\_index)

Output should look like below:

they

stumble

who

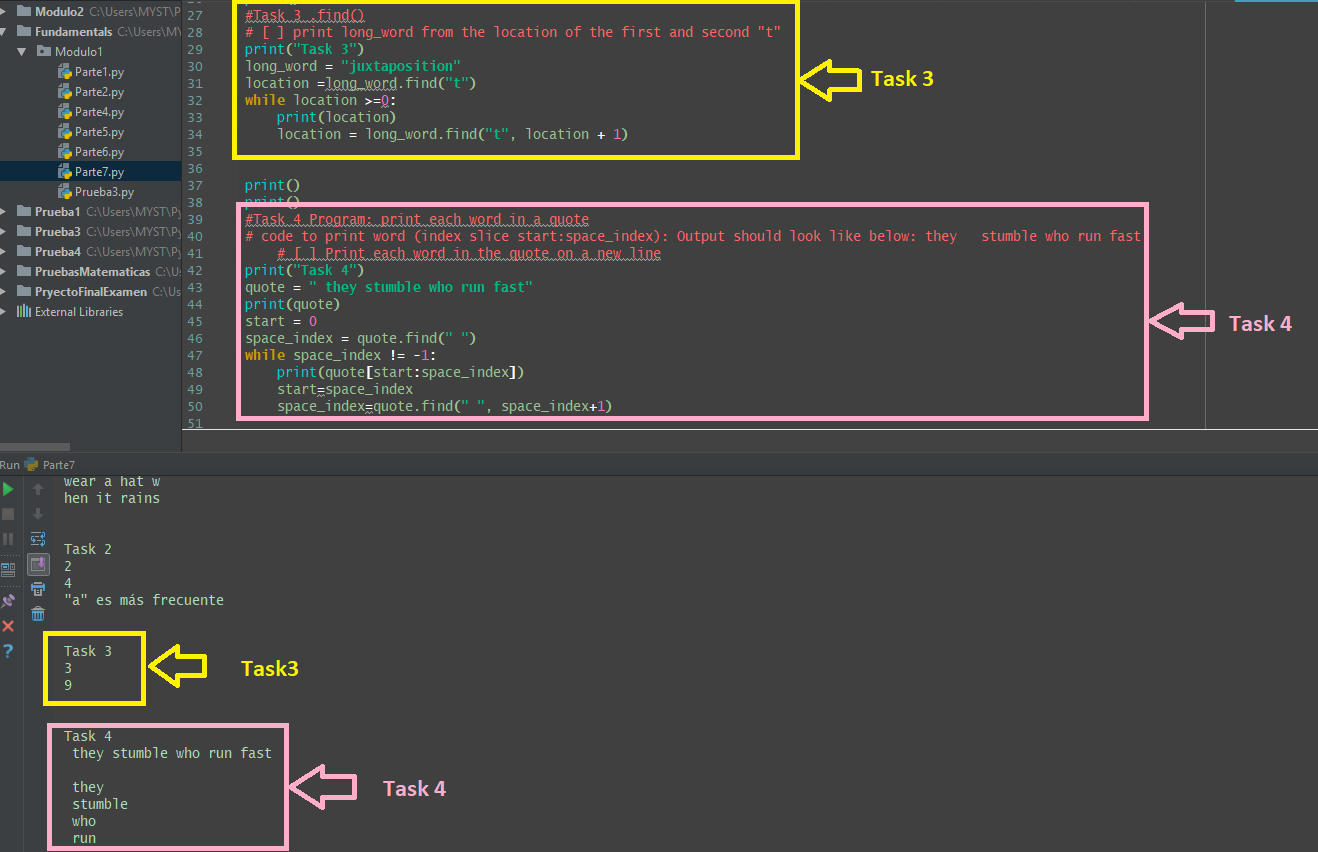
run

fast

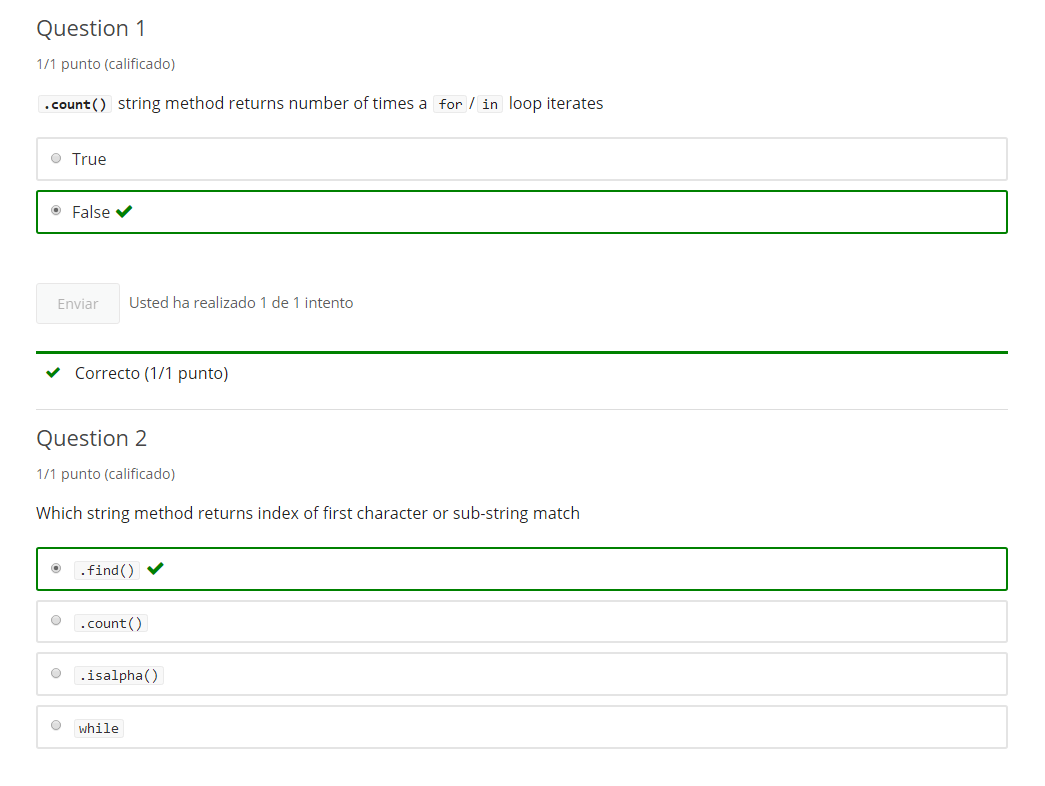
# [ ] Print each word in the quote on a new line

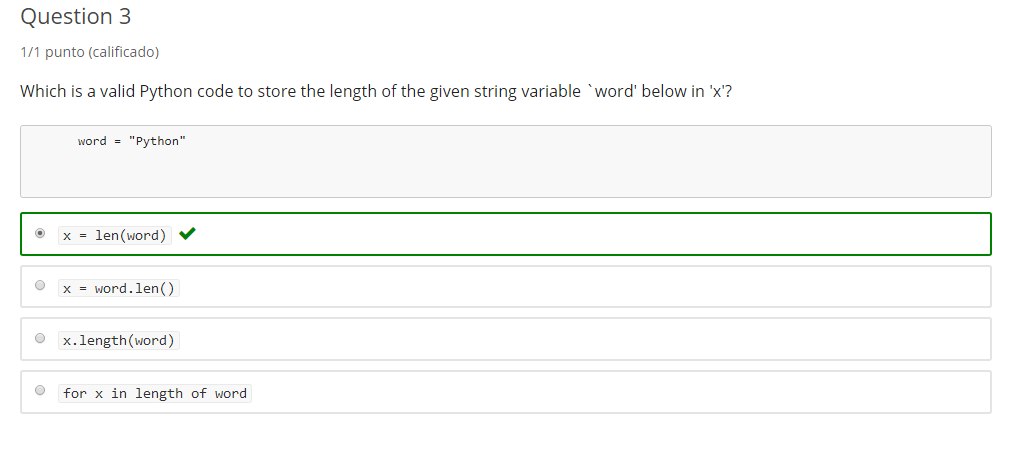
quote = "they stumble who run fast"

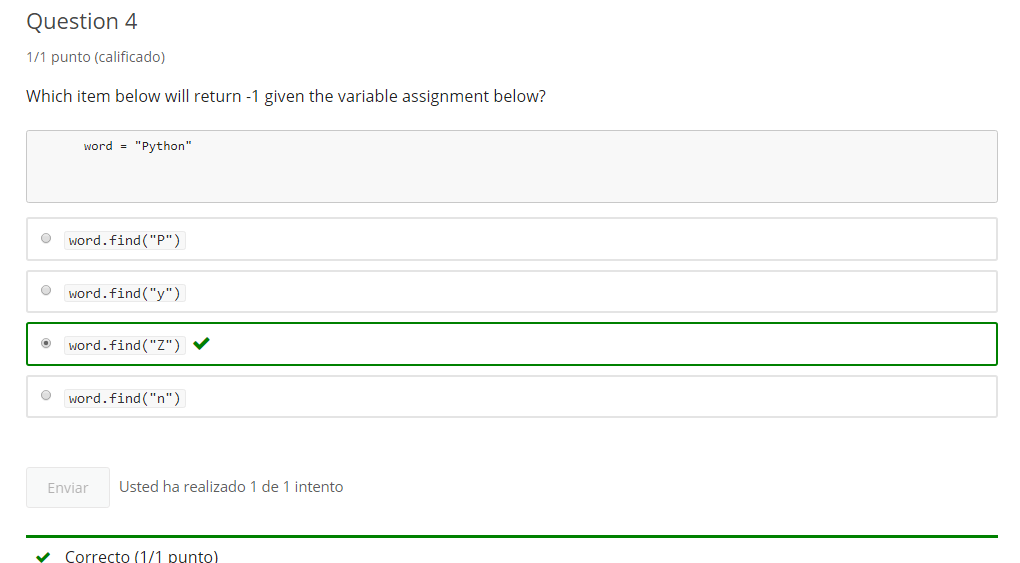




**5.3** **Self-Check: Module 1: Lesson 1.4**







**6. Module 1 Practice**

# 2-1 Intro Python Practice

Jupyter Notebook: Mod1\_2-1\_IntroPy\_Practice.ipynb

The link to the .ipynb Jupyter Notebook files are in the last lesson of section 0 of module 1

## Sequence: String

# Student will be able to

* Work with String Characters
* Slice strings into substrings
* Iterate through String Characters
* Use String  Methods

## Task 1

## Access String Characters

### working\_string[index]

# [ ] access and print the second character from planet\_name: "u"

planet\_name = "Jupiter"

# [ ] access and print the first character from planet\_name: "J"

planet\_name = "Jupiter"

# [ ] access and print the first and last characters from planet\_name

planet\_name = "Jupiter"

# [ ] using a negative index access and print the first character from planet\_name: "J"

planet\_name = "Jupiter"

## Task 2

## slice

### working\_string[start:stop]

### working\_string[start:stop:step]

# [ ] print planet\_name sliced into the first 3 characters and remaining characters

planet\_name = "Neptune"

# [ ] print 1st char and then every 3rd char of wise\_words

# use string slice with a step

wise\_words = 'Play it who opens'

# [ ] print planet\_name in reverse

## Task 3

## iterate a String

### for letter in sentence:

# [ ] Get user input for 1 fav\_food

# [ ] iterate through letters in fav\_food

# - print each letter on a new line

# [ ] iterate work\_tip string concatenate each letter to variable: new\_string

# [ ] concatenate the letter or a "-" instead of a space " "

# tip: concatenate string example: word = word + "a"

work\_tip = "Good code is commented code"

# [ ] Print the first 4 letters of name on new line

name = "Hiroto"

# [ ] Print every other letter from 2nd to last letter of name

name = "Hiroto"

## Task 4

## Program: Mystery Name

* get user input for first\_name
* create an empty string variable: new\_name
* iterate through letters in first\_name Backwards
  + add each letter to new\_name as you iterate
  + Replace the letter if "e", "t" or "a" with "?" (hint: if, elif, elif, else)
* print new\_name

**example: "Alton" = "no?l?"**

# [ ] Create Mystery Name

## Task 4

## len(), .find(), .count()

* len(working\_string)
* .find("i")
* .find("i",start)
* .find("i", start, end)
* .count("i")
* .count("i", start)
* .count("i", start, end)

# [ ] find and display the length of the string: topic

topic = ".len() returns the length of a string"

# [ ] use len() to find and display the mid\_pt (middle) index (+/- 1) of the string: topic

# note: index values are whole numbers

topic = "len() can take a sequence, like a string, as an argument"

# [ ] print index where first instance of the word "code" starts using .find()

work\_tip = "Good code is commented code"

# [ ] search for "code" in code\_tip using .find()

# [ ] search substring with substring index start= 13,end = last char

# [ ] save result in variable: code\_index

# [ ] display index of where "code" is found, or print "not found" if code\_index == -1

work\_tip = "Good code is commented code"

## Task 5

# [ ] find and report (print) number of w's, o's + use of word "code"

work\_tip = "Good code is commented code"

# [ ] count times letter "i" appears in code\_tip string

# [ ] find and display the index of all the letter i's in code\_tip

# Remember: if .find("i") has No Match, -1 is returned

code\_tip = "code a conditional decision like you would say it"

print ("code\_tip:" , code\_tip)

## Task 6

## Program: Words after "G"/"g"

Create a program inputs a phrase (like a famous quotation) and prints all of the words that start with h-z

Sample input:  
enter a 1 sentence quote, non-alpha separate words: **Wheresoever you go, go with all your heart**

Sample output:

WHERESOEVER

YOU

WITH

YOUR

HEART

* split the words by building a placeholder variable: **word**
  + loop each character in the input string
  + check if character is a letter
  + add a letter to **word** each loop until a non-alpha char is encountered
* **if** character is alpha
  + add character to **word**
  + non-alpha detected (space, punctuation, digit,...) defines the end of a word and goes to **else**
* **else**
  + check **if** word is greater than "g" alphabetically
    - print word
    - set word = empty string
  + or **else**
    - set word = empty string and build the next word

Hint: use .lower()

# [] create words after "G"

# sample quote "Wheresoever you go, go with all your heart" ~ Confucius (551 BC - 479 BC)

**7. End of Mod coding assignment**

## 1. Module 1 Required Code Description

**Video:** **Module1RequiredCodeDescriptionV13.mp4**

>> Let's walk through the module one: Required Coding Activity, which is a program called words after G. It's a program that it takes a saying and add it like a common phrase and then does a output on it with words that are starting after the letter G. Let's talk a little bit about the requirements for this page. In this course, there are specific keywords that must be used. If you do not use these, you will not get credit when you submit your solution code. So, first of all, we're going to have printed output like we saw with the words that are greater than G. We're going to use print(), and we're going to have input() as the source of those words originally, then we'll filter some of them out. We're going to use for/in, so that we're going to iterate through several items and then we're also going to use.isalpha() to test for alphabetical characters. If it's all alphabetical characters, we'll have a conditional logic, right? We'll use that.isalpha() with if, else, or you can use else elif either one. And then, also you have to use either.upper() or.lower() because when we do string comparisons, we want to make sure that upper case can be equal to a lower case by changing both those to isupper() or islower(). So, let's look at our EdX page. And we'll see there's those familiar requirements here that we just went over and some more explanation of what we're doing and a flow chart that can show us what we need to build. So let's first take a quick look at the Jupiter notebook. It has the same content as the EdX page. I've loaded the code previously. Let's see how it works. So, it asks me to enter a sentence. And so, I'm going to put in our familiar code, "Wheresoever you go, go with all your heart", and we enter that in and we see the output here. Only the letters that have the first letter that is greater than the alphabetical letter G. So, H, I, J, K, etcetera. Let's take a look at our flow chart here. We'll start and get a quote input. We're going to iterate through all the characters in the string. So, we look through and we say, is an alphabetical character. So, the first character when we went over here, we could enter in the "wheresoever". And so we're going to start with a W, and that character is alphabetical. So, we're going to add it to a word. So, we basically have the word equals a string. And so it starts off with W and then we are going to keep going through this loop here and we're going to get for each character in the string, we keep going getting strings. So then we go W, then we go H. And then we can go through all the way with our E-R-E, so ever. And then a space is going to come. But when that space comes it will not get added to word because, when we hit "No", it's not of alphabetical character, then what we're going to do is check that word that we created. So, we have this word here that we've created, wheresoever. We check the first letter, and if it's greater than equal to H, so we're going to make sure it's like either we're going to make both sides upper or lower, and then we're going to print that word. So we made that string go empty again and we're going to keep going through the letters in there. And so then next, we're going to get the word "you". And then when we hit the space we're going to print that because it starts with a letter greater than G, then the gos, both of those gos as we come through this to check with word, it's not going to be capitalized. And so what we want to do in that case is take the new path here, and just go directly to making the string empty. And you might have to have two different places where you set that to empty in your solution, but we're going to set that string to empty without printing. So, we don't go that way, so we don't print the character. So, we get the wherever, you and then the gos don't get printed and then as with and so on. So, we keep doing that. When we get to the end of the string, there's no characters left. This, when there's no more characters left, it just ends. Now, depending on how you input your string, in order to get to this comparison, if we're going to check the word, we have to get a non-alphabetical character to go this way or we're going to have to just say, okay, here we're going to have to put that check, the same check logic. We're going to have to put it over here. So, you can decide if somehow you want to ensure that there's a character at the end here somehow or you want to get through here. That will make more sense as you run through. If you see that the last word does not get printed, then you'll see that okay, because there's no space afterwards, we're not going this way and not getting to that word. So, if you run into that, you can come back this video and say, why am I not getting here? And that's because you need a space on the end there. And so you should handle that as the programmer. Let's go look about how we're going to submit our code. So, when you're in EdX, you see those tabs there. And then for the program assignment, we're going to go to the submission. And when you submit the code, we have a somewhat primitive submission device in that you're not able to see how it's formatted when you paste it in here. But what you need to do is go into your notebook. And so after you write all of your solution code in the cell, you're going to copy that entire cell, you're going to copy that. And then back to this tab and you're going to paste it in there. So, in Windows it's Ctrl+C, Ctrl+V. So, from the one to tab to the other, so you're going to copy from the notebooks and paste it into EdX. And then you're going to submit that. And then if you used all the words then it will show up green. And it shows you used your one of two attempts. Otherwise, it will show the one of two of attempts and be red. And if that is the case, you need to look back to make sure that you had each of the keywords used, and your code needs to be running as well. All right, let's code.

# Module 1 Required Coding Activity

Introduction to Python (Unit 2) Fundamentals

**This Activity is intended to be completed in the jupyter notebook, Required\_Code\_MOD1\_IntroPy.ipynb, and then pasted into the assessment page that follows.**

The link to the .ipynb Jupyter Notebook files are in the last lesson of section 0 of module 1

This is an activity from the Jupyter Notebook **Practice\_MOD01\_IntroPy.ipynb** which you may have already completed.

| **Important Assignment Requirements** |
| --- |
| **NOTE:** This program **requires** **print** output and using code syntax used in module 1 such as keywords **for**/**in** (iteration), **input**, **if**, **else**, **.isalpha()** method, **.lower()** or **.upper()** method |

## Program: Words after "G"/"g"

Create a program inputs a phrase (like a famous quotation) and prints all of the words that start with h-z

Sample input:  
enter a 1 sentence quote, non-alpha separate words: **Wheresoever you go, go with all your heart**

Sample output:

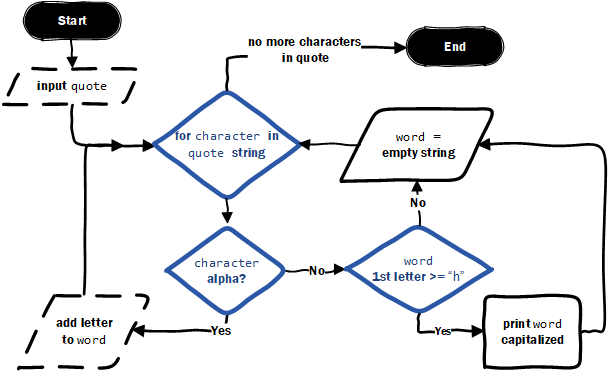
WHERESOEVER

YOU

WITH

YOUR

HEART



* split the words by building a placeholder variable: **word**
  + loop each character in the input string
  + check if character is a letter
  + add a letter to **word** each loop until a non-alpha char is encountered
* **if** character is alpha
  + add character to **word**
  + non-alpha detected (space, punctuation, digit,...) defines the end of a word and goes to **else**
* **else**
  + check **if** word is greater than "g" alphabetically
    - print word
    - set word = empty string
  + or **else**
    - set word = empty string and build the next word

Hint: use .lower()

Consider how you will print the last word if it doesn't end with a non-alpha character like a space or punctuation?

# [] create words after "G"

# sample quote "Wheresoever you go, go with all your heart" ~ Confucius (551 BC - 479 BC)

# Important: [How to submit code by pasting](https://courses.edx.org/courses/course-v1:Microsoft+DEV274x+2T2017/wiki/Microsoft.DEV274x.2T2017/paste-code-end-module-coding-assignments/)

