Module 4 - Nesting and Loops

**1. Nested Conditionals**

**1.1 Intro Python**

Jupyter Notebook: MOD04\_1-6.1\_Intro\_Python.ipynb

**Nested Conditionals**

* **Nested Conditionals**
* Escape Sequence print formatting "\"

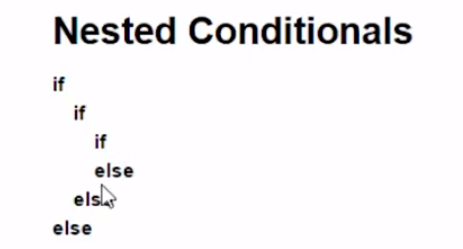
**Student will be able to**

* **create nested conditional logic in code**

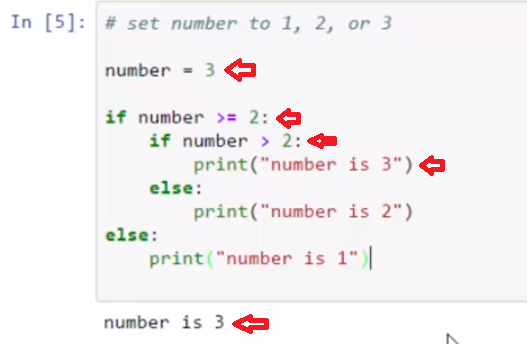
**1.2 Concept: nested conditionals**

**Video:** **ConceptNestedConditionalsV1.mp4**

**Nested conditionals allows subdecisions to be made by indenting conditional statements underneath other conditional statements.** Let's take a look at nested conditionals indenting. If we run this first if statement and it evaluates to true, the code indented will run. In this case, we see further if statements. If the second indented if statement evaluates to true, then the code underneath that will run. This means we can run a series of questions.



Let's look at the example below. We have a number that we assume will either be 1, 2 or 3 for the code below. The first if statement checks is the number greater than or equal to 2. Starting number with 1, it is not greater or equal to 2. And so the if statement moves to the matching else statement and prints the number as 1. Let's run the code. If we set the number to 2, we expect to enter the code under the first if statement because 2 is greater than or equal to 2. The second if statement says is the number greater than 2? No. So the else statement runs and it prints the "number is 2". With 3, we get the number 3 by being greater than 2 and then greater than 2 again.



We can also review this flow chart that is included in the notebook**. Here we start with the question whether we want a cheese or veggie sandwich. If you want a cheese sandwich, then you can continue to ask another question. So this, if it's true, we can go to this nested if statement and then ask the type of cheese.** Let's look at the code. Here we ask for input whether we want cheese or veggie. And we check, if it's s for cheese then we can run the code below and ask the type of cheese. If I ask for cheese, then I could say I want manchego cheese. And then it gives me my manchego cheese sandwich.



I can run the code again. I can ask for a veggie sandwich. And it gives me my vegetable special. Nested if conditionals allow us to make subdecisions in our code.

# Concept: nested conditionals

## Using nested conditionals

**if**  
**if**  
**if**  
**else**  
**else**  
**else**

### Making a sandwich

Taking a sandwich order starts with sandwich choices:

**Cheese or Veggie special?**  
if the response is **"Cheese"** "nest" a sub ask:

**Manchego or Cheddar?**

| **Nested  \*\*if\*\*  statement flowchart** |
| --- |
| Image of sandwich order flowchart |

# Example

**TIP:** click in input box before typing input

# simplified example

# [ ] review the code then run and following the flowchart paths

# \*\*\*TIP:\*\*\* click in input box before typing

sandwich\_type = input('"c" for Cheese or "v" for Veggie Special: ')

if sandwich\_type.lower() == "c":

# select cheese type

cheese\_type = input('"c" for Cheddar or "m" for Manchego: ')

if cheese\_type.lower() == "c":

print("Here is your Cheddar Cheese sandwich")

else:

print("Here is your Manchego Cheese sandwich")

else:

print("Here is your Veggie Special")

# full example: handling some invalid input and elif statement

# [ ] review the code then run following the flowchart paths including \*\*invalid responses\*\* like "xyz123"

# \*\*\*TIP:\*\*\* click in input box before typing

print("Hi, welcome to the sandwich shop. Please select a sandwich.")

sandwich\_type = input('"c" for Cheese or "v" for Veggie Special: ')

# select sandwich type sandwich\_type = input('"c" for Cheese or "v" for Veggie Special: ')

print()

if sandwich\_type.lower() == "c":

# select cheese type

print("Please select a cheese.")

cheese\_type = input('"c" for Cheddar or "m" for Manchego: ')

print()

if cheese\_type.lower() == "c":

print("Here is your Cheddar Cheese sandwich. Thank you.")

elif cheese\_type.lower() == "m":

print("Here is your Manchego Cheese sandwich. Thank you.")

else:

print("Sorry, we don't have", cheese\_type, "choice today.")

elif sandwich\_type.lower() == "v":

print("Here is your Veggie Special. Thank you.")

else:

print("Sorry, we don't have", sandwich\_type, "choice today.")

print()

print("Goodbye!")

# Task 1

## Nested if

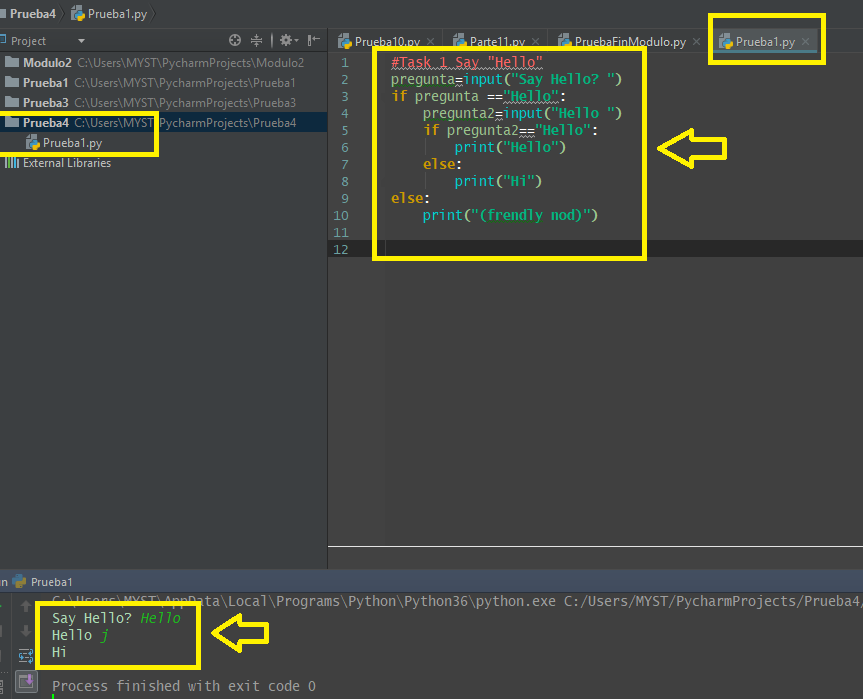
### [ ] Program: Say "Hello"

* using nested **if**

| **Say "Hello" flowchart** |
| --- |
| Image: Say "Hello" flowchart |

# [ ] Say "Hello" with nested if

# [ ] Challenge: handle input other than y/n



# Task 2

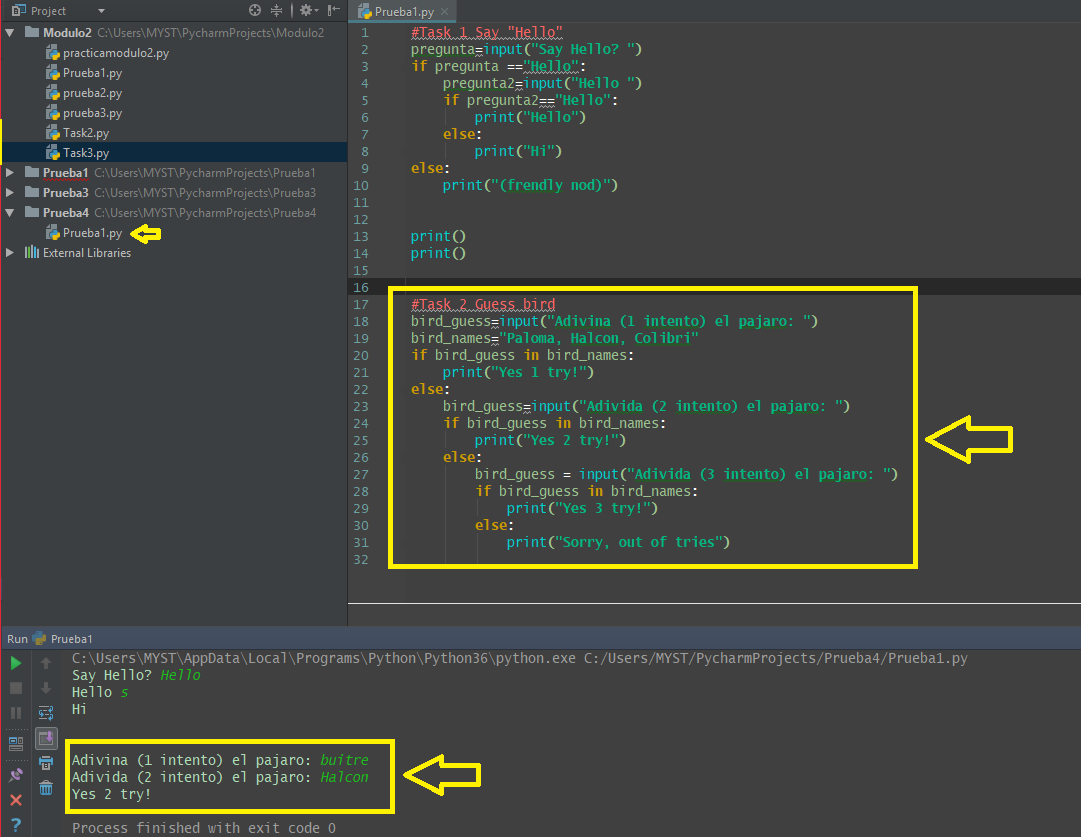
## Nested if - testing for False

### Program: [ ] 3 Guesses

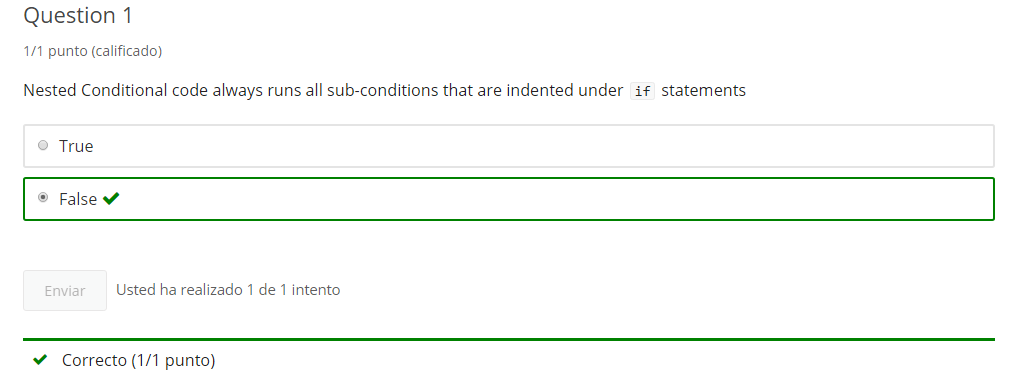
* use nested if statements complete the flowchart code
* create a **bird\_names** string variable with the names of 1, 2, 3 or more birds to make it easier
* get **bird\_guess** input and use **bird\_guess in bird\_names** to generate Boolean True/False
* if the the guess is wrong (**False**) create a sub test until the user has had 3 guesses

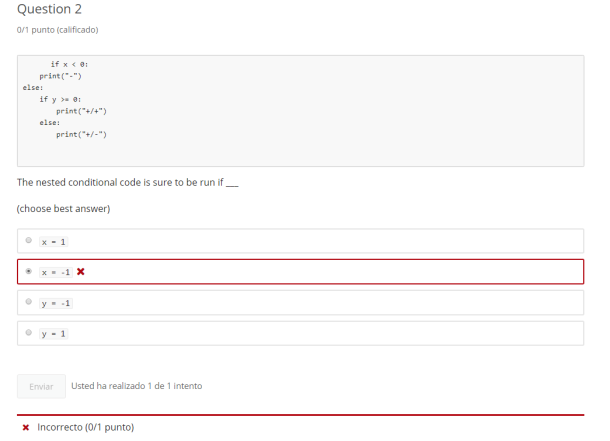
| **3 Guesses ("Guess the Bird") flowchart** |
| --- |
| Image of Guess the Bird flowchart |

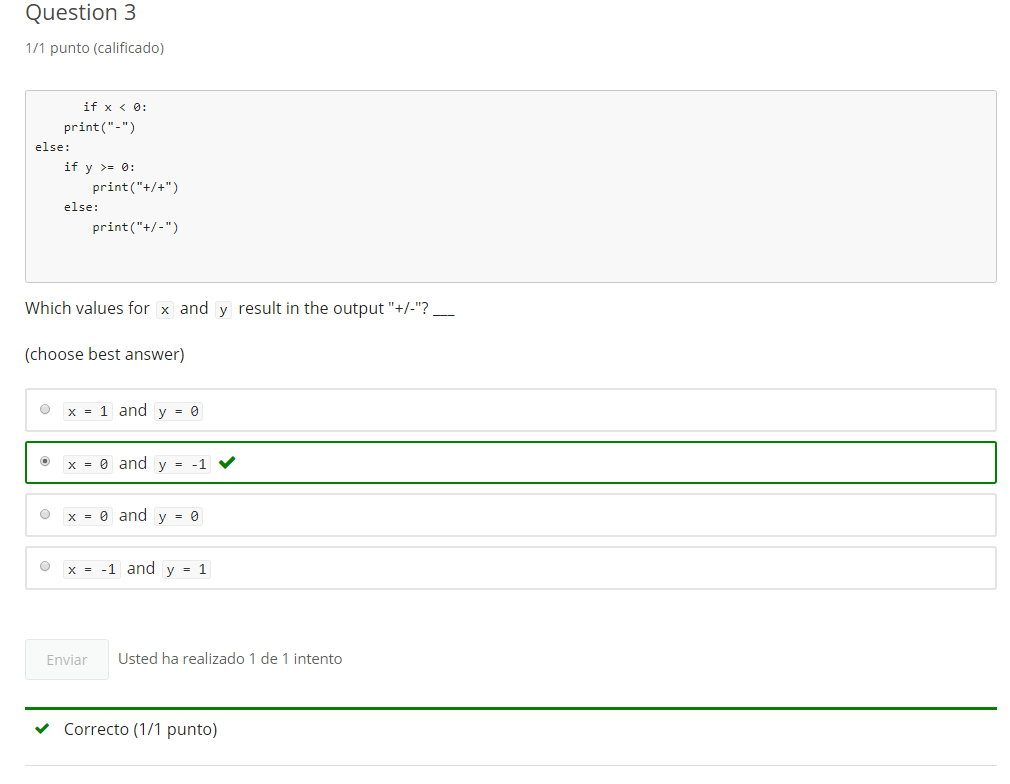
# [ ] Create the "Guess the bird" program

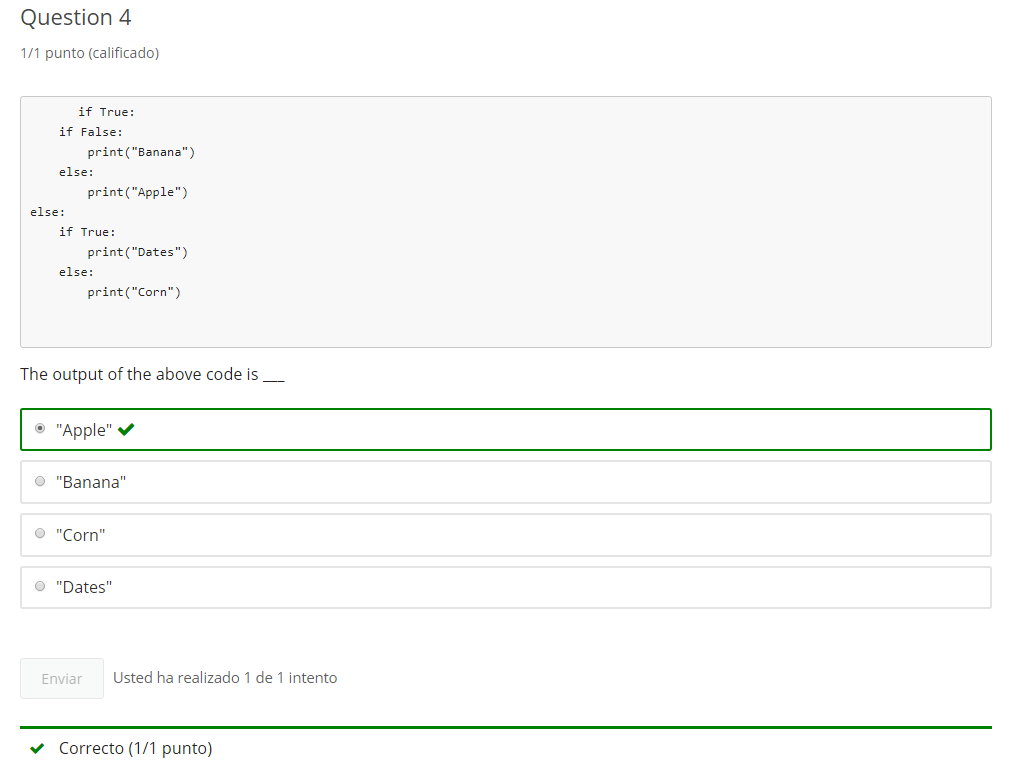


**1.3 Self-Check: Module 4 - Section 1**









**2. Escape Sequences**

**2.1 Intro Python**

Jupyter Notebook: MOD04\_1-6.2\_Intro\_Python.ipynb

**Nested Conditionals & escape sequence**

* nested conditionals
* **print formatting with the () escape sequence**

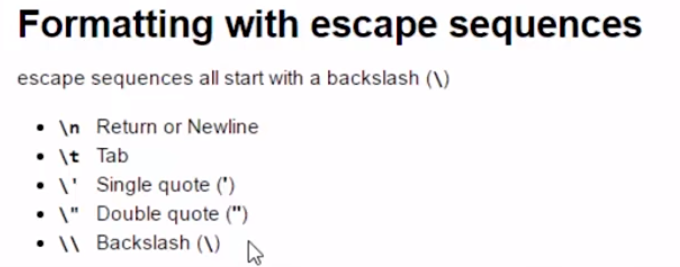
**Student will be able to**

* **print format print using escape sequence ()**

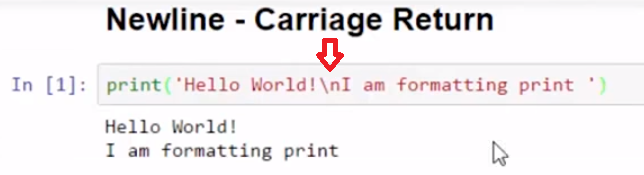
**2.2 Escape Sequences**

**Video:** **EscapeSequenceV2.mp4**

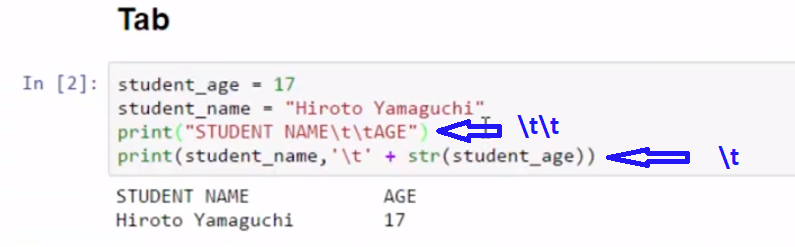
**Python uses the backslash as and escape sequence. This allows Python to display quotes, backslashes, tabs and new lines. >> Our formating character examples all start with a backslash.**



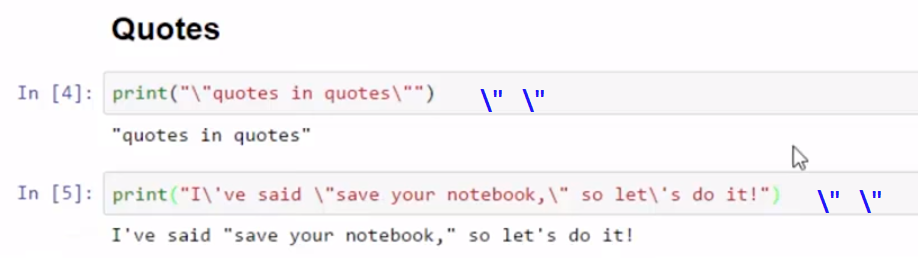
**Let's start with the Newline Carriage Return formatting character, \n. This creates a new line in the middle of a string.** So, Hello World is followed by a new line and I am formatting print.



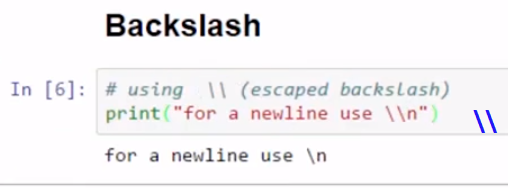
**The Tab character, \t Creates a tab in the middle of a string. Here, I have two tabs to help create a header.** Let's run the code. We see the two tabs, and then another between the name and the age.



**To display quotes, I use a backslash, and then either a double or single quote(\"). In this example, I want to show a display of quotations around the text without putting the single quotes around them. Where this becomes more useful is when I have a long string, and I want both a single quote, or an apostrophe, and double quotes.** The backslash quote, in either single or double, will display the characters. Here is my apostrophe or single quote and the double quotes.



**Sometimes, we just want to show the backslash character. Use two backslashes \\. Double backslash results in a backslash being printed.**



We can use formatting characters that all start with the escape sequence backslash to help format our output.

# Concept: Escape Sequence

## Formatting strings using escape sequences

### Escape Sequences

* escape sequences all start with a backslash (**\**)
* escape sequences can be used to display characters in python reserved for formatting
  + **\\**   Backslash (**\**)
  + **\'**   Single quote (**'**)
  + **\"**   Double quote (**"**)
* escape sequences are part of special formatting characters
  + **\t**   Tab
  + **\n**   return or newline

We use escape sequences in strings - usually with print() statements

# Example

# review and run example using \n (new line)

print('Hello World!\nI am formatting print ')

# review and run code using \t (tab)

student\_age = 17

student\_name = "Hiroto Yamaguchi"

print("STUDENT NAME\t\tAGE")

print(student\_name,'\t' + str(student\_age))

# review and run code

# using \" and \' (escaped quotes)

print("\"quotes in quotes\"")

print("I\'ve said \"save your notebook,\" so let\'s do it!")

# using \\ (escaped backslash)

print("for a newline use \\n")

# Task 1

Format using backslash (**\**) escape sequences

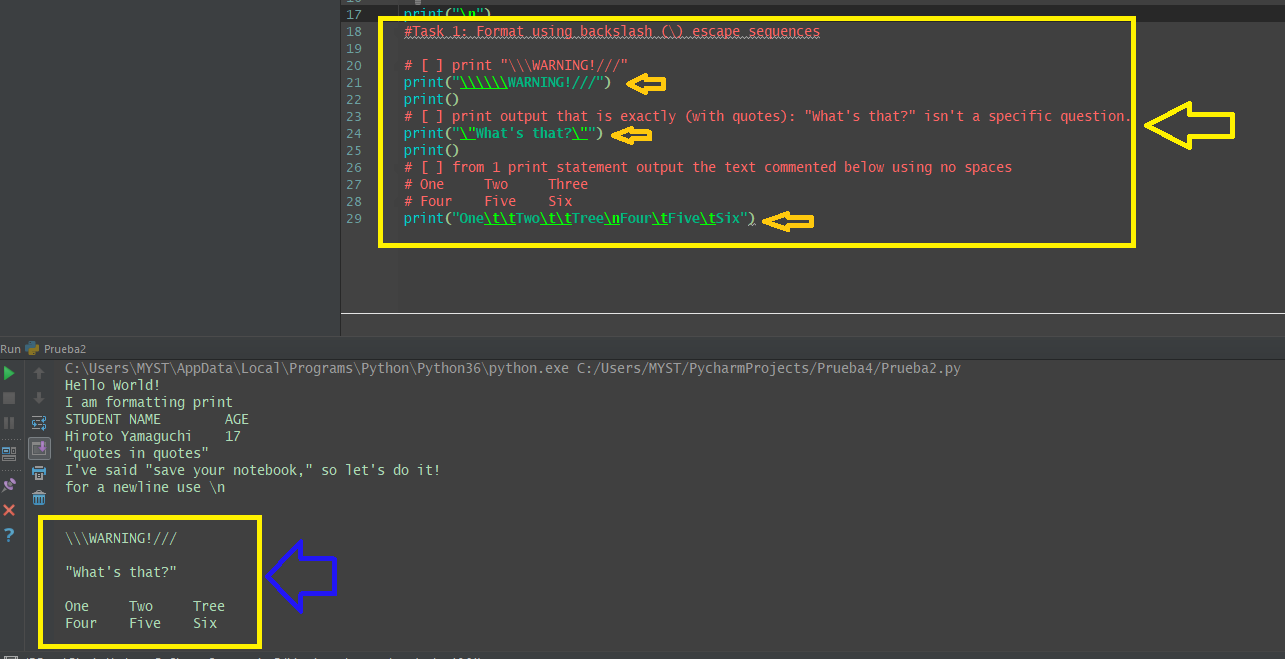
# [ ] print "\\\WARNING!///"

# [ ] print output that is exactly (with quotes): "What's that?" isn't a specific question.

# [ ] from 1 print statement output the text commented below using no spaces

# One Two Three

# Four Five Six



# Task 2

## Program: pre\_word() Function

Function has a single string parameter that it checks s is a single word starting with "pre"

* Check if word starts with "pre"
* Check if word .isalpha()
* if all checks pass: return **True**
* if any checks fail: return **False**
* **Test**
  + get input using the directions: \*enter a word that starts with "pre": \*
  + call pre\_word() with the input string
  + test **if** return value is **False** and print message explaining not a "pre" word
  + **else** print message explaining is a valid "pre" word

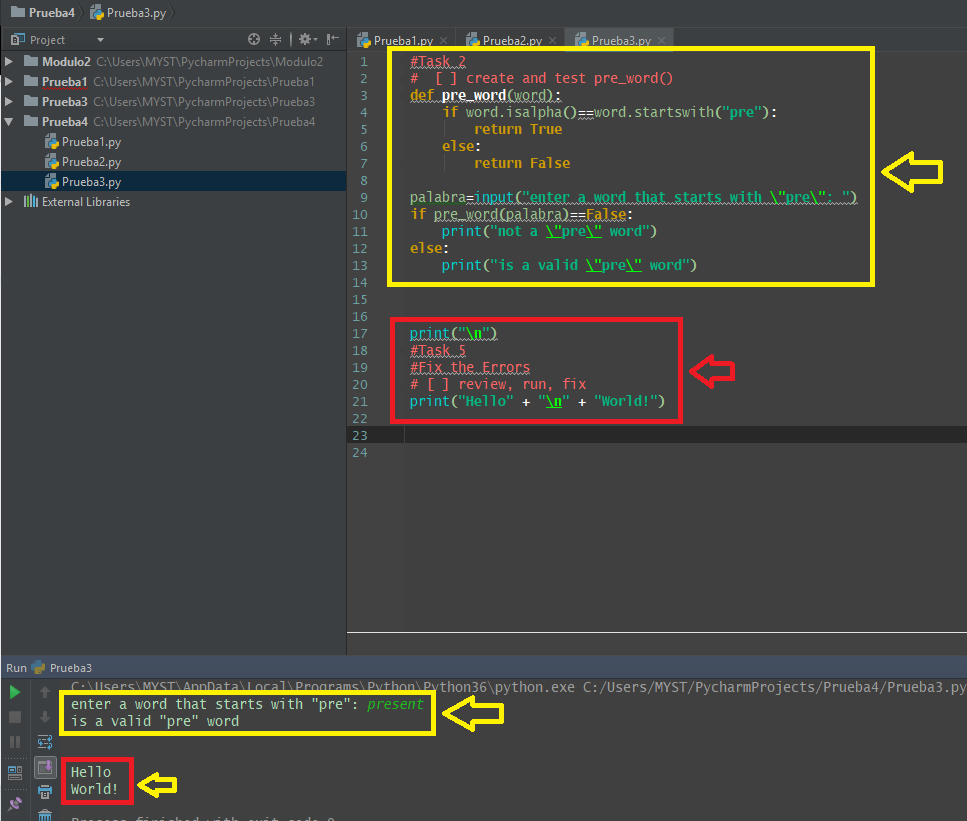
# [ ] create and test pre\_word()

# Task 5

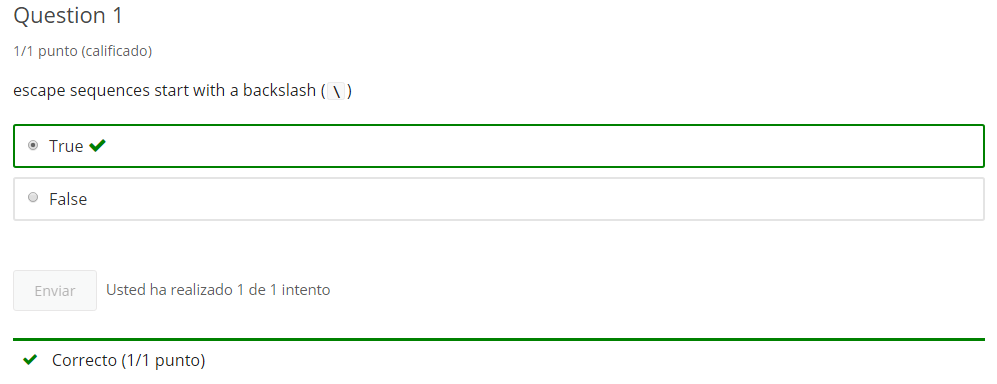
## Fix the Errors

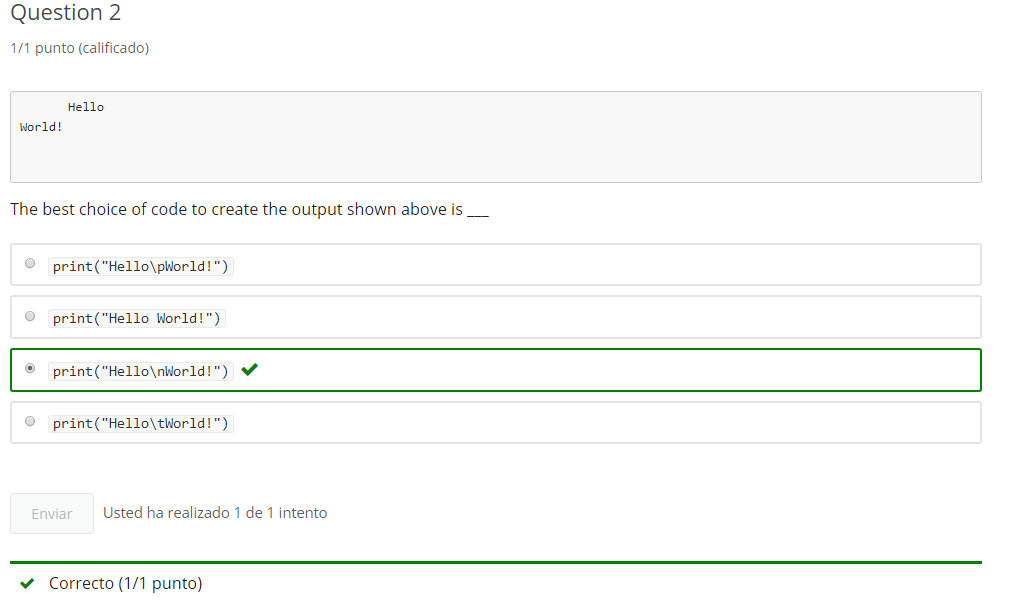
# [ ] review, run, fix

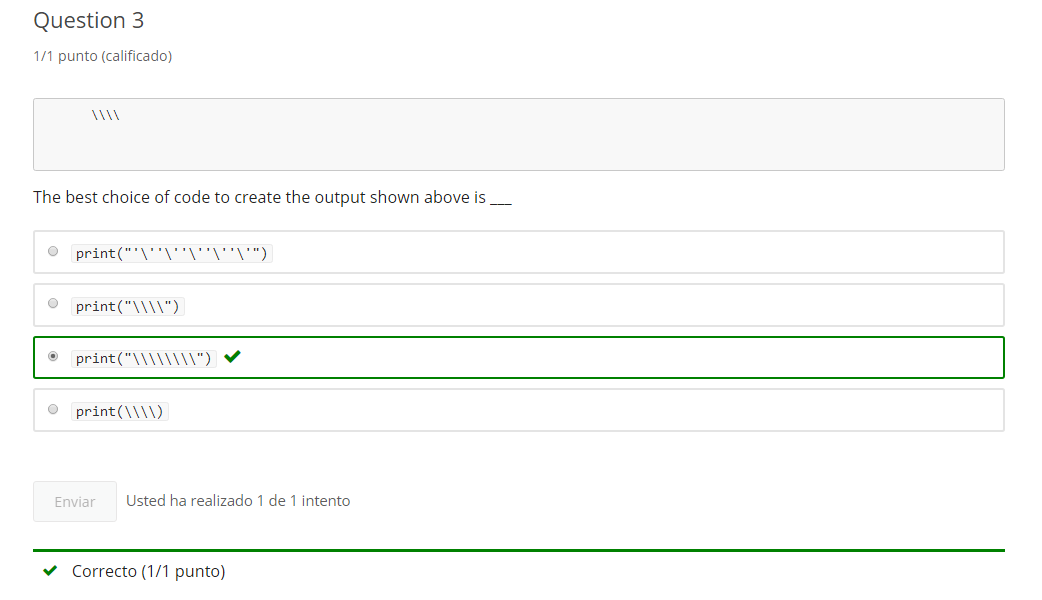
print("Hello" + \n + "World!")

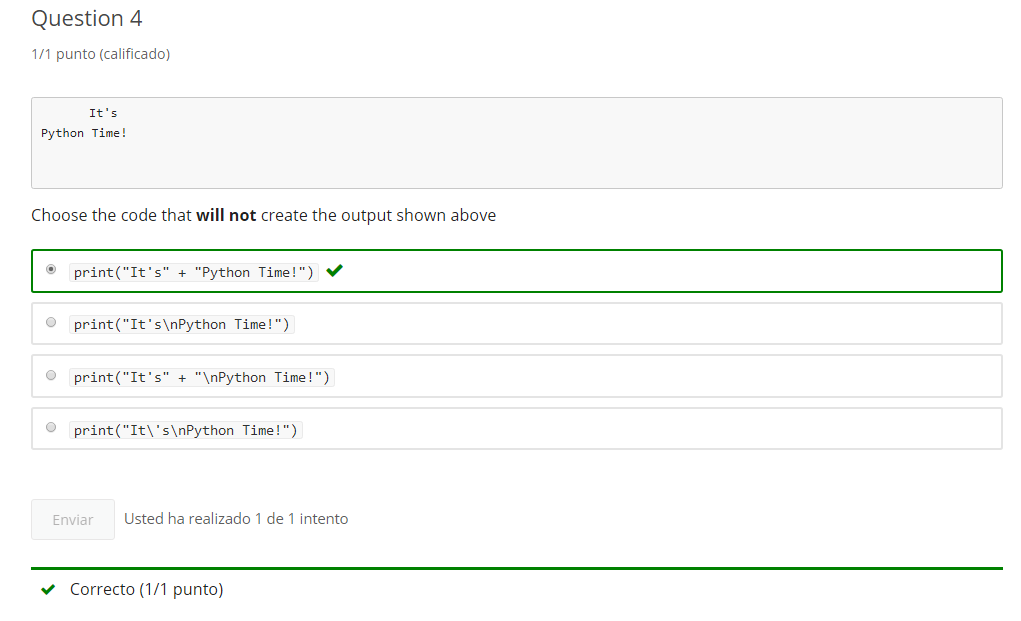


**2.3 Self-Check: Module 4 - Section 2**









**3. 'while' loop and incrementing**

**3.1 Intro Python**

Jupyter Notebook: MOD04\_1-7.1\_Intro\_Python.ipynb

while() loops & increments

* **while True or forever loops**
* **incrementing in loops**
* Boolean operators in while loops

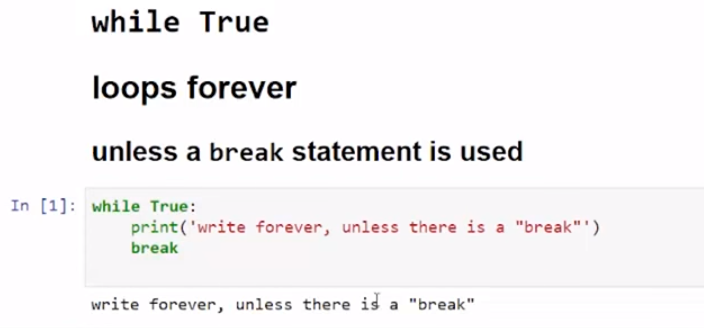
**Student will be able to**

* **create forever loops using while and break**
* **use incrementing variables in a while loop**

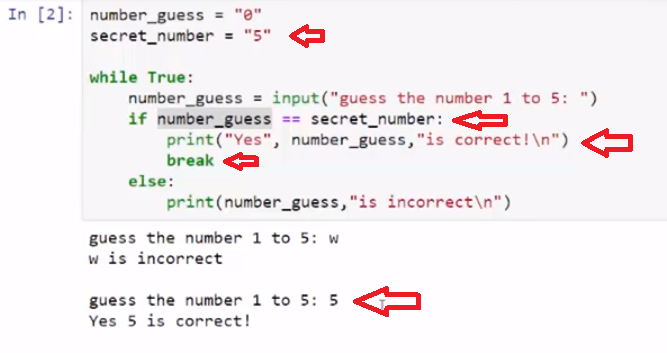
**3.2 while True loop**

**Video:** **whileTrueLoopV3.mp4**

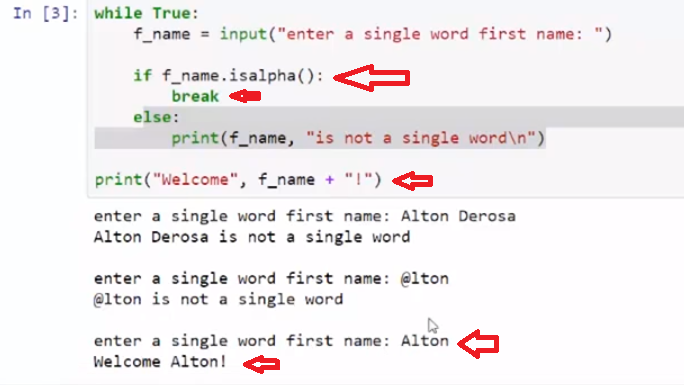
**A while loop runs while a condition is true. If you write while with the keyword true, it will run forever. And is what is known as an infinity loop.** Let's take a look at why the while True loop runs forever. **Here, we have the keyword true. It can't evaluate to anything but true. And so this while loop keeps continuing when it gets to the end of the indentation. But this while true loop only runs one time, because we have the keyword break.** **Break breaks out of a loop.** **It returns to the code after the indentation.** And would run anything that follows. Let's run the code here. It runs once and then breaks. It's kind of a meaningless while true loop, because it's hard-coded break and there's no conditional.



So we need to look at an example that has a conditional for the break, an If statement. **Here, if this evaluates to true, a break statement will run. This while true loop takes a number\_guess input. It has a secret number that it's gonna compare against. The number is 5, hard coded above. Let's run the code. If I put in anything but a number, it will be incorrect. Because we're comparing my number guess from the input to the string five.** If it's not the number five in a string format, then it will give the message that your guess is incorrect. So we can put in the number. And finally yes we are correct.



Let's look at another example, where we get input and we ask it to be just a single word. **So a single word has no spaces and it is a word and uses only alphabetical characters. So we're gonna use the is alpha check. This will evaluate as false for anything that is not an alphabetical character.** So a space or some type of symbol. I'll put in a full name. It says that Alton Derosa is not a single word. Because the space failed the isalpha check, the else statement runs. I can't spell Alton with the at symbol. That is not an alphabetical character, so again, it fails, and the else runs. Welcome, Alton. We finally wrote just a single word of all alphabetical characters.



A while true loop will loop forever, until it encounters a break statement.

# Concept: while True loop

## Using the while True: loop

**while True:** is known as the **forever loop** because it ...loops forever

Using the **while True:** statement results in a loop that continues to run forever  
...or, until the loop is interrupted, such as with a **break** statement

## break

### in a while loop, causes code flow to exit the loop

a **conditional** can implement **break** to exit a **while True** loop

# Example

## while True loops forever unless a break statement is used

# Review and run code

# this example never loops because the break has no conditions

while True:

print('write forever, unless there is a "break"')

break

# [ ] review the NUMBER GUESS code then run - Q. what cause the break statement to run?

number\_guess = "0"

secret\_number = "5"

while True:

number\_guess = input("guess the number 1 to 5: ")

if number\_guess == secret\_number:

print("Yes", number\_guess,"is correct!\n")

break

else:

print(number\_guess,"is incorrect\n")

# [ ] review WHAT TO WEAR code then run testing different inputs

while True:

weather = input("Enter weather (sunny, rainy, snowy, or quit): ")

print()

if weather.lower() == "sunny":

print("Wear a t-shirt and sunscreen")

break

elif weather.lower() == "rainy":

print("Bring an umbrella and boots")

break

elif weather.lower() == "snowy":

print("Wear a warm coat and hat")

break

elif weather.lower().startswith("q"):

print('"quit" detected, exiting')

break

else:

print("Sorry, not sure what to suggest for", weather +"\n")

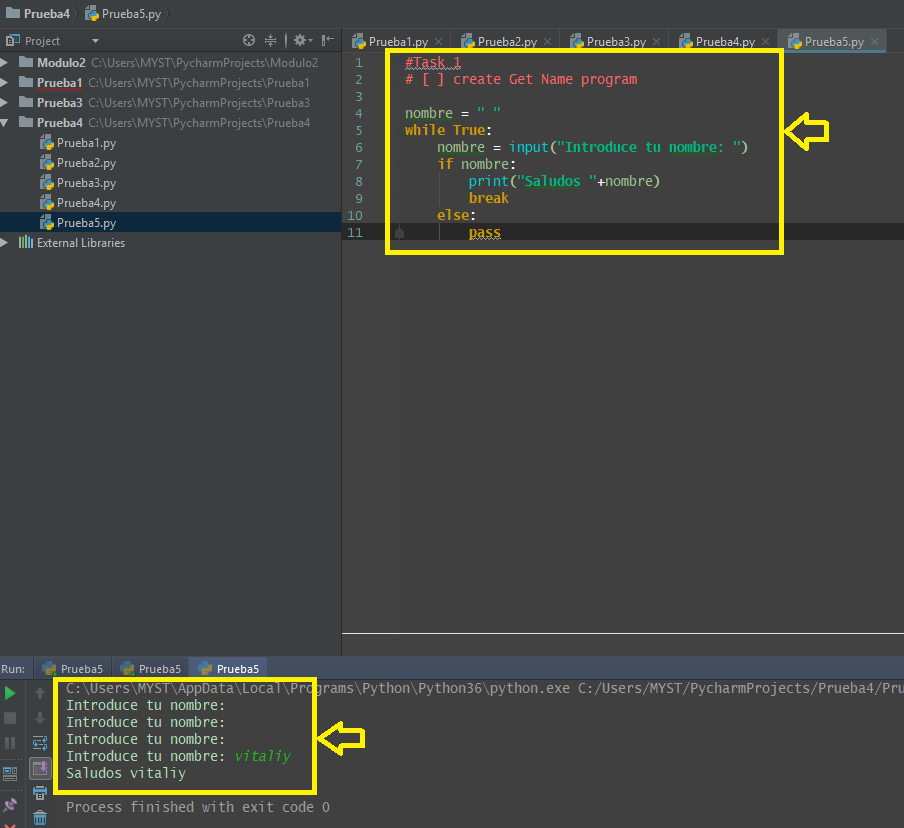
# Task 1

## while True

### [ ] Program: Get a name forever ...or until done

* create variable, familar\_name, and assign it an empty string (**""**)
* use **while True:**
* ask for user input for familar\_name (common name friends/family use)
* keep asking until given a non-blank/non-space alphabetical name is received (Hint: Boolean string test)
* break loop and print a greeting using familar\_name

# [ ] create Get Name program

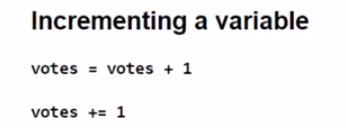


**3.3 Increment a variable**

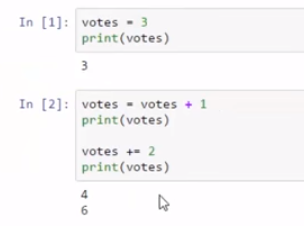
**Video:** **IncrementVariableV4.mp4**

**Incrementing a variable is a common task that we do inside of a loop, where we wanna count each time a loop executes.**

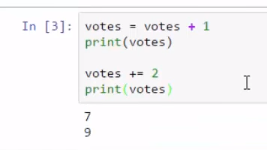
To use a counting variable in a loop where we want to increment by 1, we can use the following code. The variable votes is set equal to itself + 1. Each time this code runs, the votes tally is increased by 1. Shorthand uses this format. votes += 1, which means votes equals itself + 1.



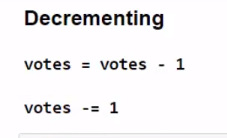
Let's run some code, where we set votes = 3. And then we want to increment it by 1, using the votes = votes + 1 method. And then the shorthand, we're gonna increment it by 2. When the code runs, we see that the votes tally increased from 3 to 4 and then 2 more to 6.



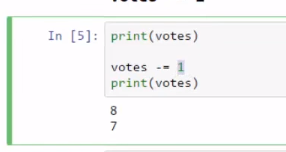
We can run the code again. Now it moved from 6 to 7 and then 2 more to 9.



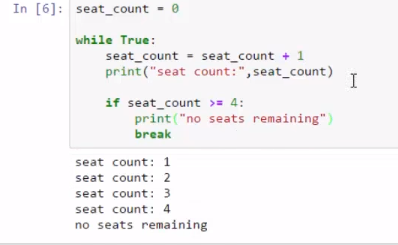
To decrement, we use the same format. Votes equals itself but now we decrease by 1, we subtract. The shorthand uses the votes -=1. Votes equals itself- 1.



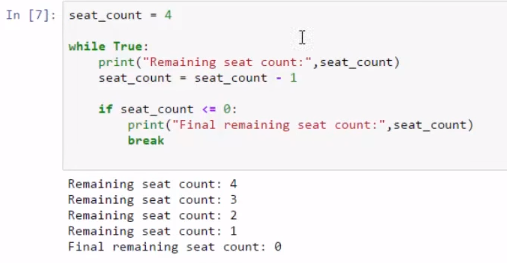
Let's print out the current value of votes and decrement by 1. We can run the code again. Decrement will decrease the value and in this case, we can count down by 1. Here, we're going to keep count of the available seats.



We start with 0, then we run a while True loop and we count and add 1 seat each time seat\_count equals itself + 1. If the seat\_count is equal to, or greater than 4, then we say we're out of seats. Let's run the code. Here we see the count incrementing 1, 2, 3, 4, and then breaking from the loop.



Another use of a while True loop is counting with a decrement. We start with 4 seats available. And then we count as each seat is filled, decrementing by 1. So, seat count equals itself- 1. When the seat count is equal to or less than zero, then we say we are out of seats and our final count is 0. Let's run the code. So we start with 4 and then we decrement until we have 0 seats available.



A variable can be incremented or decremented by setting it equal to itself + or- 1.

# Concept: Variable Increment

## Incrementing a variable

## Incrementing

### votes = votes + 1     or   votes += 1

## Decrementing (negative incrementing)

### votes = votes - 1     or   votes -= 1

# Example

# [ ] review and run example

votes = 3

print(votes)

votes = votes + 1

print(votes)

votes += 2

print(votes)

print(votes)

votes -= 1

print(votes)

# [ ] review the SEAT COUNT code then run

seat\_count = 0

while True:

print("seat count:",seat\_count)

seat\_count = seat\_count + 1

if seat\_count > 4:

break

# [ ] review the SEAT TYPE COUNT code then run entering: hard, soft, medium and exit

# initialize variables

seat\_count = 0

soft\_seats = 0

hard\_seats = 0

num\_seats = 4

# loops tallying seats using soft pads vs hard, until seats full or user "exits"

while True:

seat\_type = input('enter seat type of "hard","soft" or "exit" (to finish): ')

if seat\_type.lower().startswith("e"):

print()

break

elif seat\_type.lower() == "hard":

hard\_seats += 1

elif seat\_type.lower() == "soft":

soft\_seats += 1

else:

print("invalid entry: counted as hard")

hard\_seats += 1

seat\_count += 1

if seat\_count >= num\_seats:

print("\nseats are full")

break

print(seat\_count,"Seats Total: ",hard\_seats,"hard and",soft\_seats,"soft" )

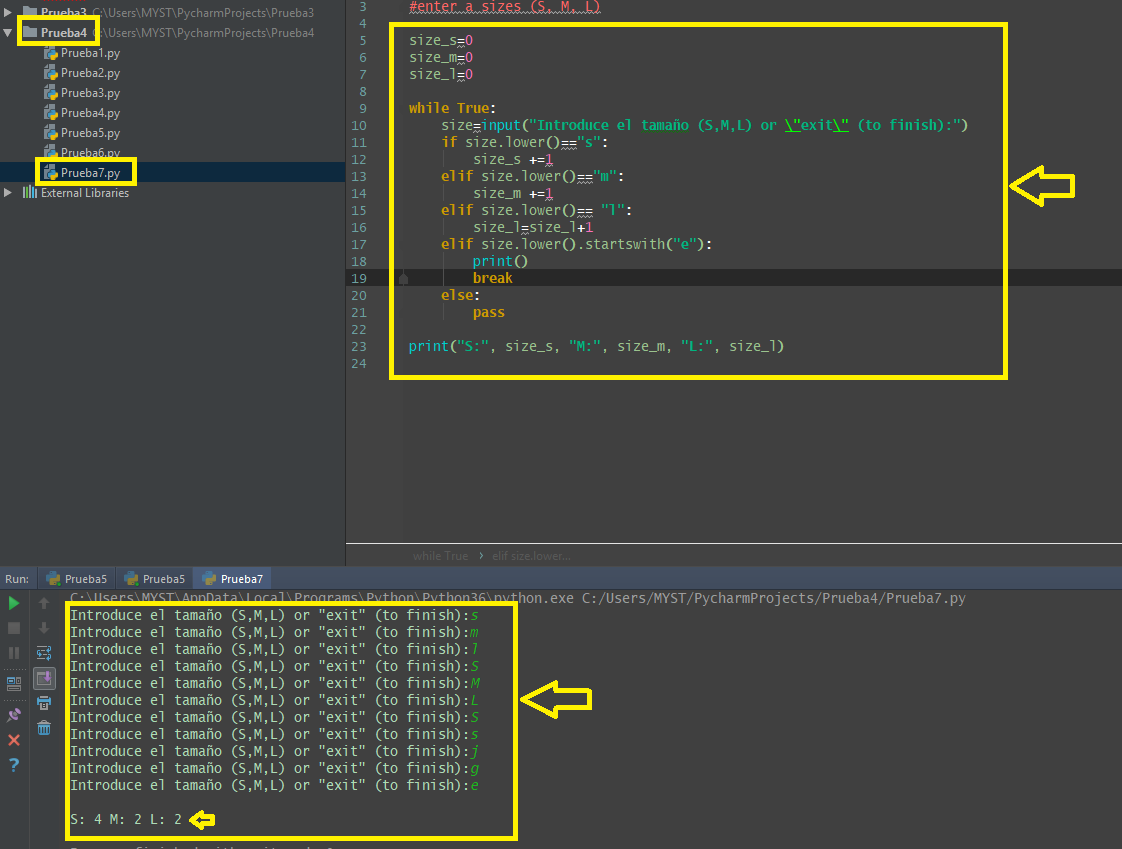
# Task 2

## incrementing in a while() loop

### Program: Shirt Count

* enter a sizes (S, M, L)
* tally the count of each size
* input "exit" when finished
* report out the purchase of each shirt size

# [ ] Create the Shirt Count program, run tests

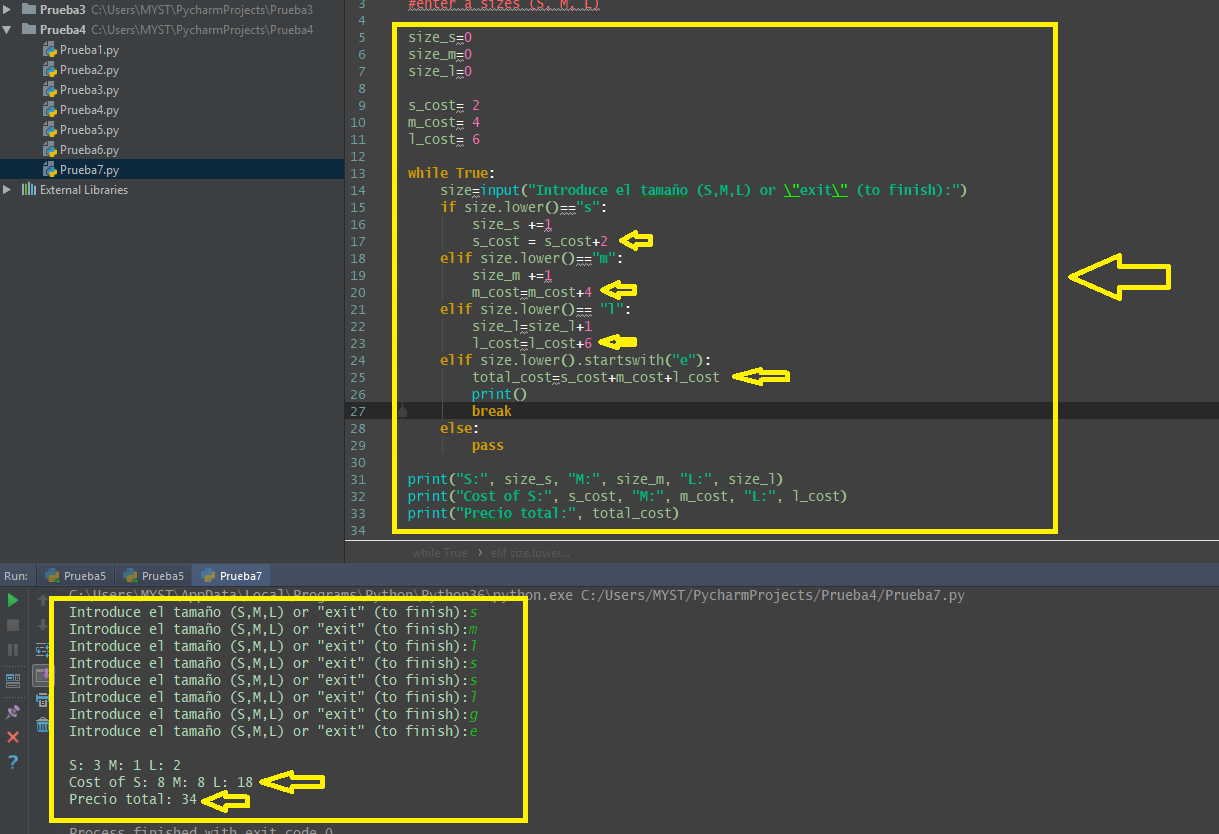


### CHALLENGE: Shirt Register (optional)

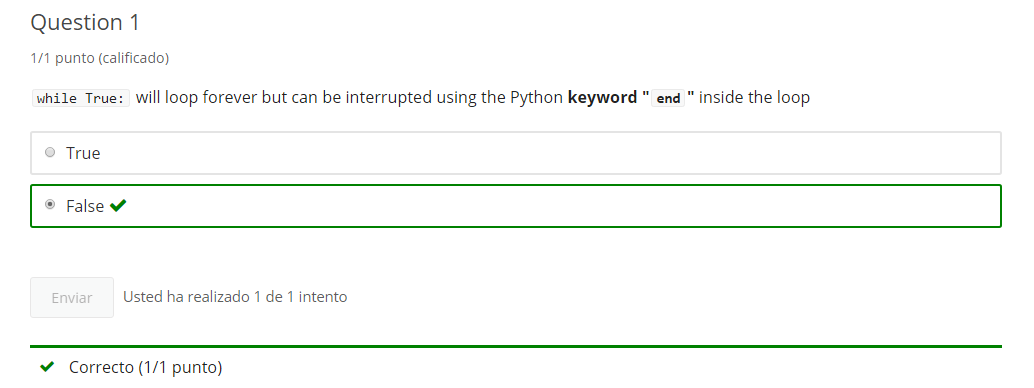
Update the **Shirt Count** program to calculate cost

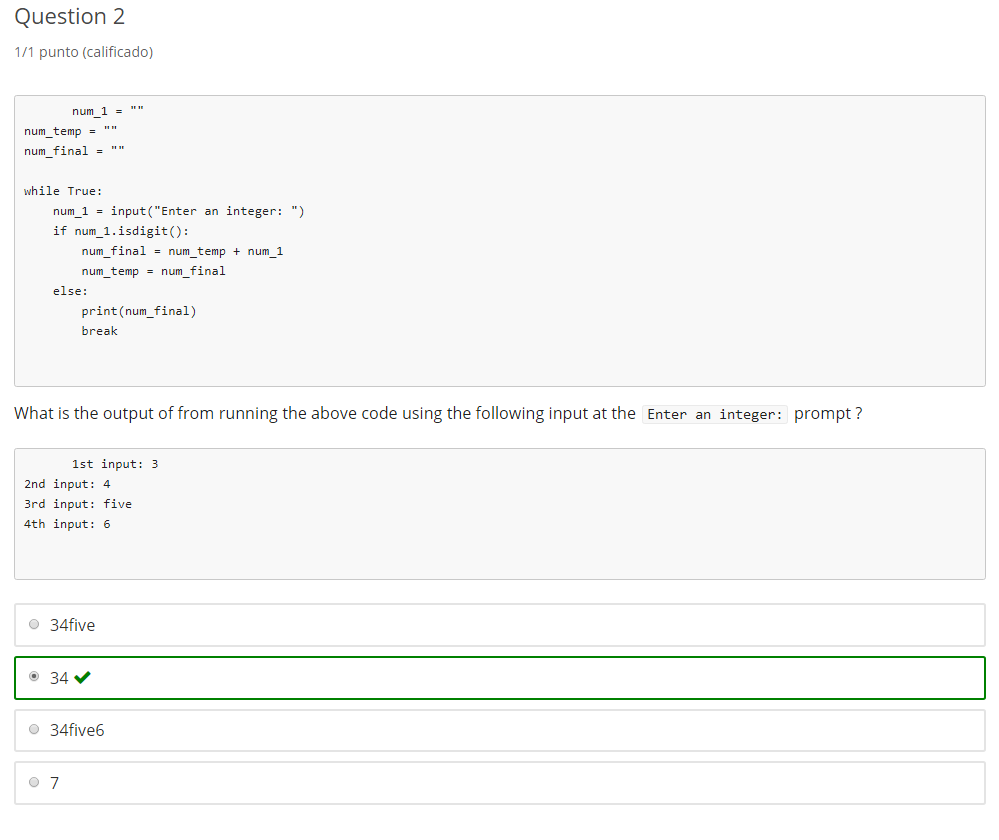
* use shirt cost (S = 6, M = 7, L = 8)
* to calculate and report the subtotal cost for each size
* to calculate and report the total cost of all shirts

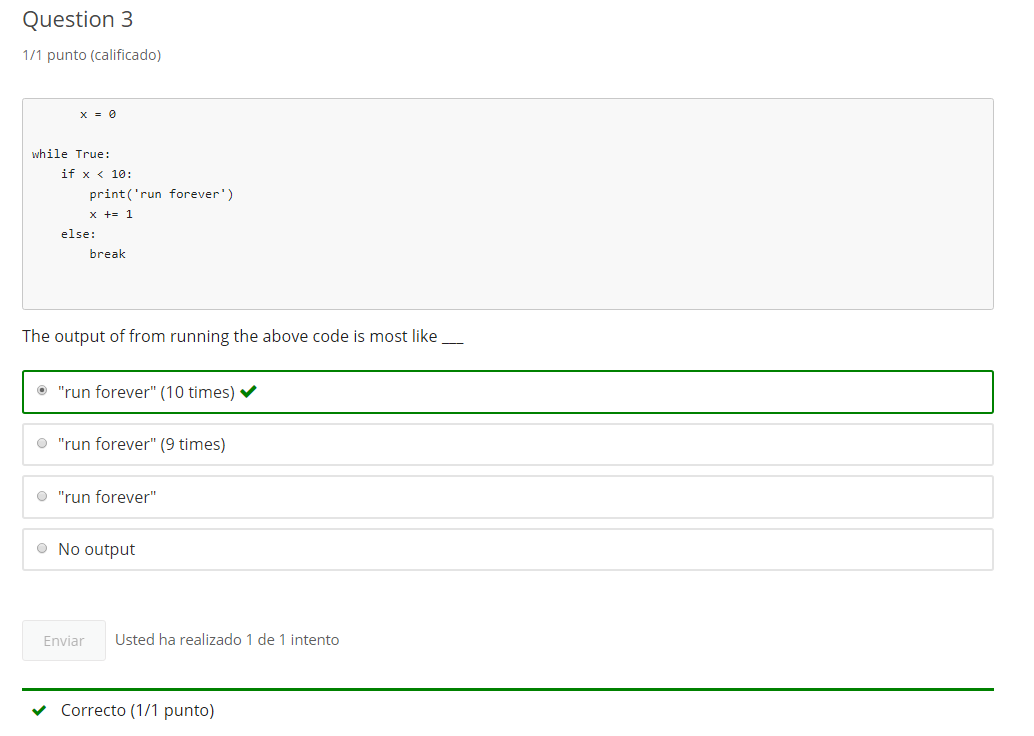
# [ ] Create the Shirt Register program, run tests

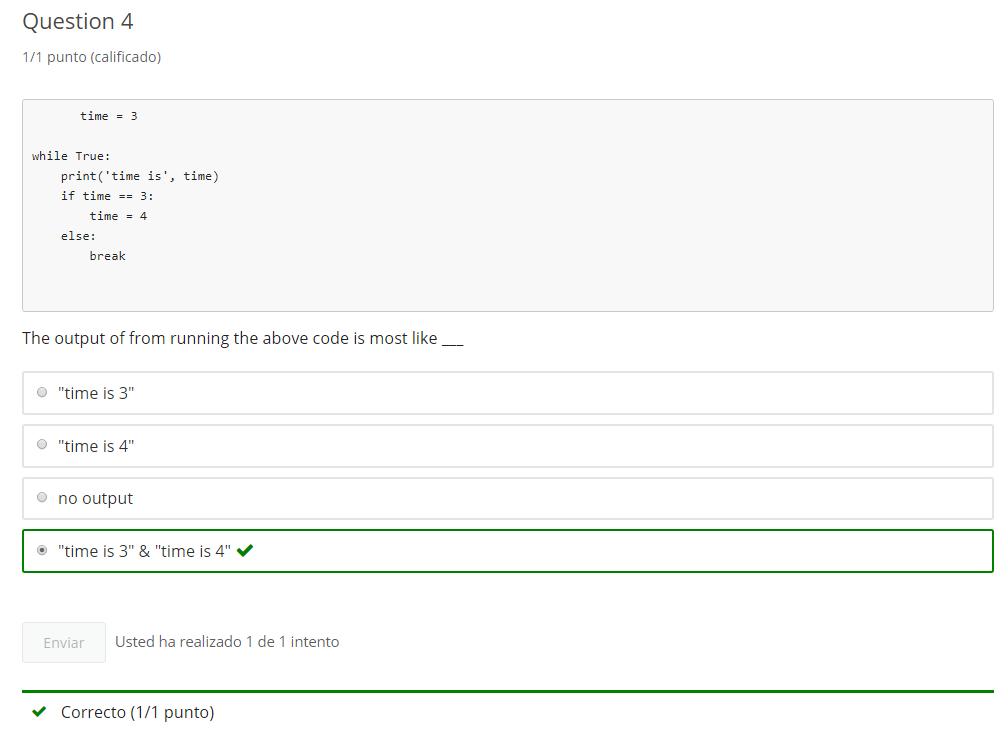


**3.4 Self-Check: Module 4 - Section 3**









**4. 'while' Boolean loops**

**4.1 Intro Python**

Jupyter Notebook: MOD04\_1-7.2\_Intro\_Python.ipynb

while() loops & increments

* while True or forever loops
* incrementing in loops
* **Boolean operators in while loops**

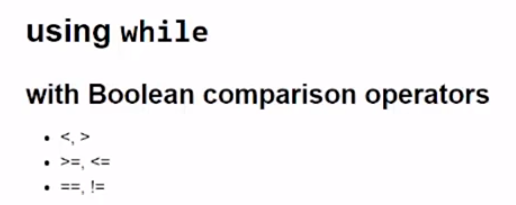
**Student will be able to**

* **control while loops using Boolean operators**

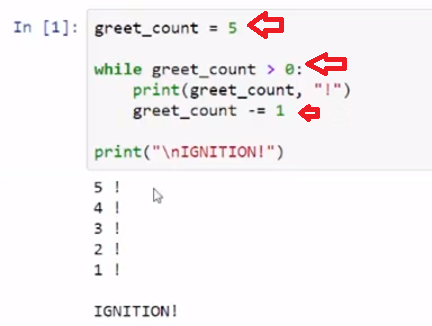
**4.2 Concept: while loop with Boolean comparisons**

**Video:** **WhileLoopWithBooleanComparisonsV5.mp4**

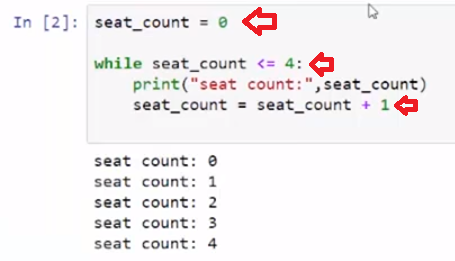
**While loops can be controlled with Boolean comparison operators so that we can stop the loop when a match is made or a number goes over a certain value.** Let's review our comparison operator so we can use them in a while loop. We have < and >. We have >= and <=. ==, !=.



**In our first example, we'll run a while loop while a value is greater than 0. We start with a greet\_count = 5 and it will print out that number. Then it will decrement that by one. So it will decrease it by one each time it runs through the loop, until it is no longer greater than zero.** Let's run the code. So we see it started with 5 and prints all the way til one. When it gets to 1, then we see that one minus one will be 0, and it's no longer greater than 0. And the loop exits to this point outside of the indentation, and runs ignition.



In our final example, we run a while loop checking if the seat\_count <= 4. Here, we see that there's a print statement and an increment of the seat\_count value as we go through the loop, starting with the seat\_count to find at 0. Let's run the code. **Here we see the increment running and this comparison operator, when it hits 4, the loop ends.** Code ends up outside of the indentation and that is the code.



We can control the looping of a while statement by using Boolean comparison operators.

# Concept: while loop with Boolean comparisons

## while while with Boolean comparison operator

### while x < 5:

with increment we used **break** when count becomes greater than some number

The same result can be achieved by using **while x < 5:**

# Example

# review and run GREET COUNT

greet\_count = 5

# loop while count is greater than 0

while greet\_count > 0:

print(greet\_count, "!")

greet\_count -= 1

print("\nIGNITION!")

# Task 1

## while with comparison operator

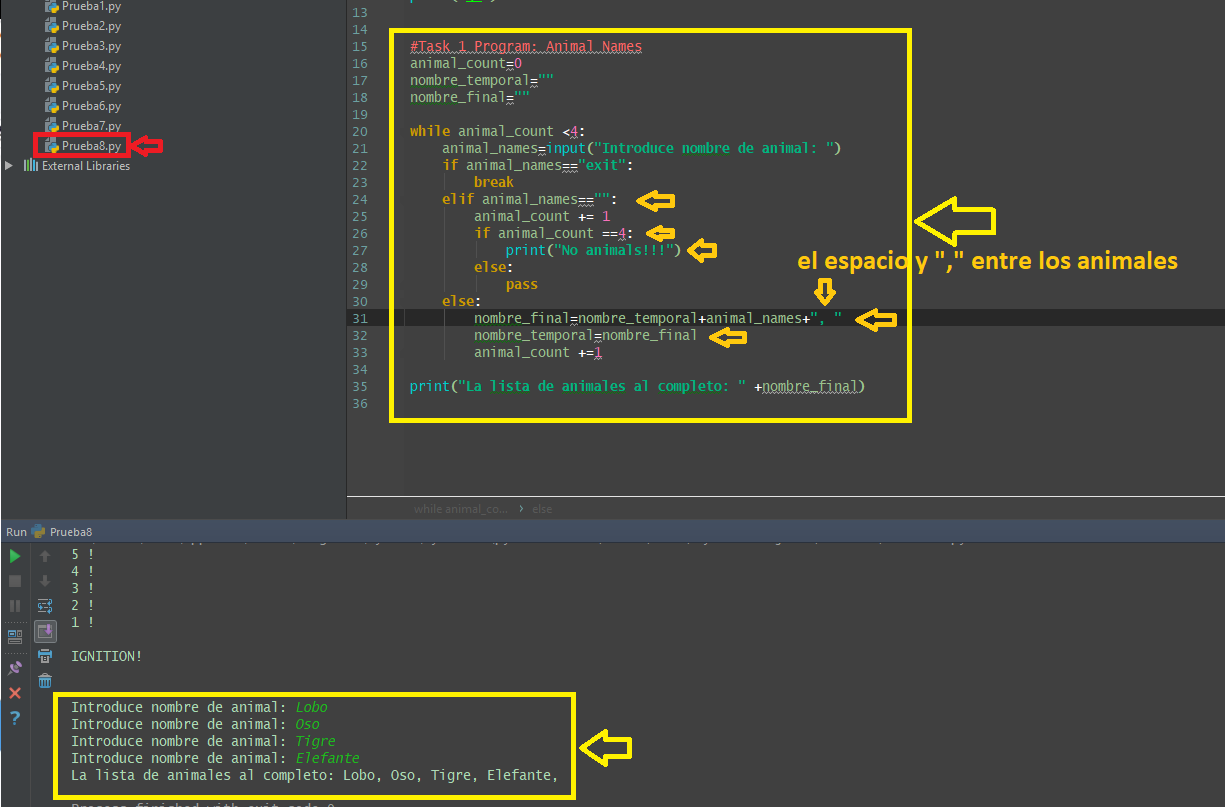
### Program: Animal Names

* Use **while** to get the user input, animal\_name, of 4 animals
  + use a counter, num\_animals, in the while loop condition
  + append the names to a string variable, all\_animals
  + User can exit early by typing "exit" (check if animal\_name is "exit" and break)
* when the loop finishes, print the names of all\_animals

-bonus: print "no animals" if animal\_name is empty after exiting the loop

**Tip:** Think about Sequence and variables that need to be initialized before the while loop

# [ ] Create the Animal Names program, run tests

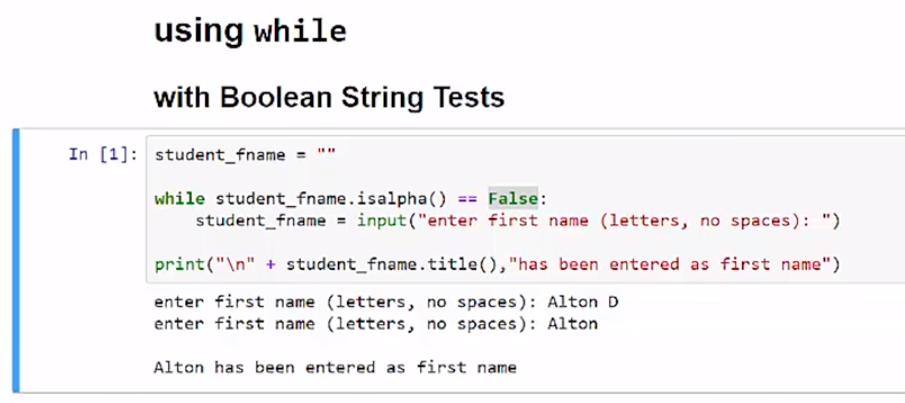


**4.3 while with Boolean string tests**

**Video:** **whileWithBooleanStringTestsV6.mp4**

**A while loop can be used with any boolean, but it's common to use it with a boolean string test to make sure that our input is in the format that we want.**

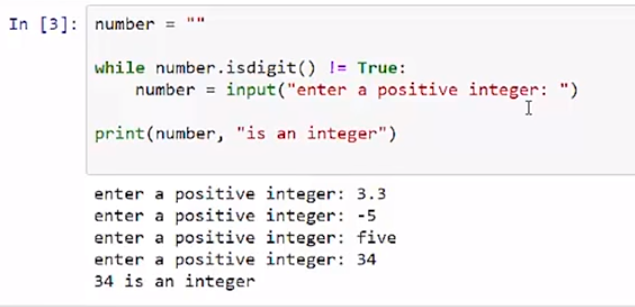
An example of a format that we might want is just a first name. And so in the first example here**, we define a variable called student fname, standing for first name, and just give it an empty string. And then we have a boolean string test in a while loop. Here, we're gonna check if the student first name is alphabetical characters. And if that is false, then we're gonna run the code to get input for the name because once it's true then we have a alphabetical character name and then we can print it out with this statement below.** Let's run this code. If I put in Alton D, That fails, a fail here of this test is equal to false. And so the code runs again and gets input. **The problem here is that a space is not an alphabetical character.** So we just looking for a first name, and although this isn't always true that first name don't have spaces, just for the purposes of this task we want to illustrate that. So now I can just enter Alton with no spaces. Then I get my greeting, Alton has been entered as first name.



**In this second example, we have a while loop that wants all of the input in uppercase. We just did the same thing we did in the last loop where we check if this is false. If the message is upper, is returned false then we keep going in the loop. And the way that we compared it this time is we said, it's not equal to true. And so the case is when I put in high and lowercase, that's false and that is not equal to true. So when that happens, then this code below will be run. So I'll keep doing the input until I go in uppercase**, so let's run that. Sorry, I can't hear, please yell a message. Can't hear, yell a message. So now the message is received.



**Finally, there's an example where we want a number input, and we say enter a positive integer. This is a case where isdigit works very well. Isdigit will return as false if you enter a decimal point or a minus sign, and so only number, numeric characters can be entered to have this be true.** As long as is number returns false, then it is not true and the code below runs. So let's run that code. I'm gonna go 3.3, -5, and even a number written in text. Finally, I'll just put in 34, which is a positive integer.



**We can use while loops to keep running through at input to get the proper format that we need for our code.**

# Concept: while loops with Boolean string tests

## Using while with a Boolean String Tests

A while loop can be controlled by Boolean strings such as while name.isalpha() == False:

# Example

# review and run example that loops until a valid first name format is entered

student\_fname = ""

while student\_fname.isalpha() == False:

student\_fname = input("enter student\'s first (Letters only, No spaces): ")

print("\n" + student\_fname.title(),"has been entered as first name")

# Task 2

## Using while with a Boolean String

### Program: Long Number

#### Create variables

* **int\_num** and get user input **string** of only digits
* **long\_num** and initialize it as an empty string

#### Create a while loop that runs as long as the input is all digits

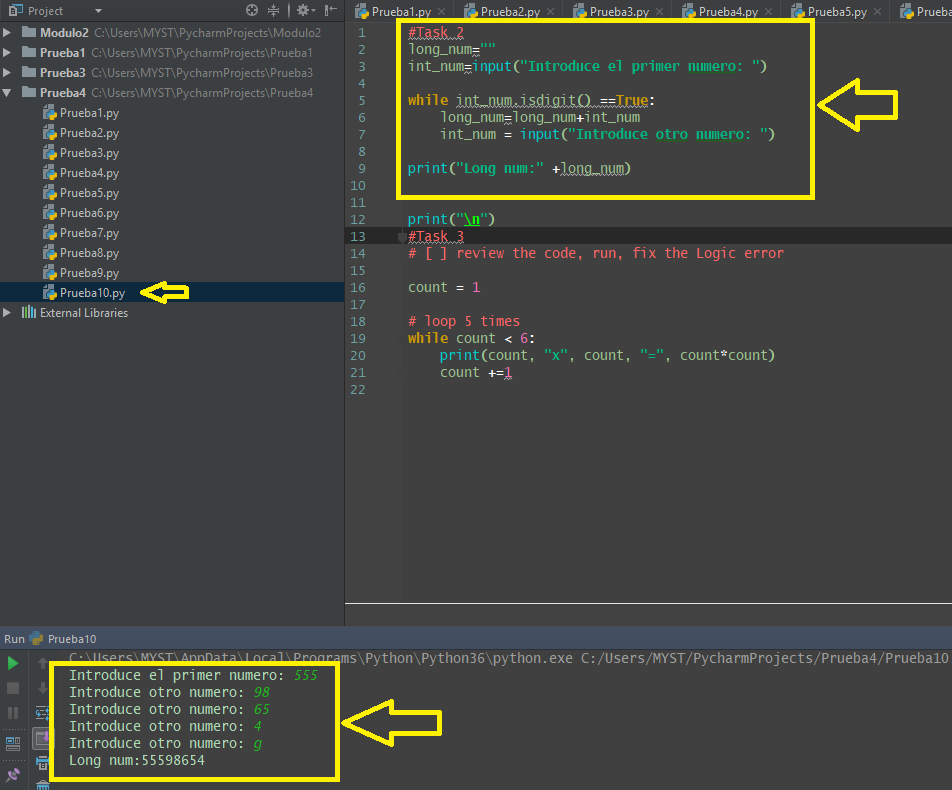
#### Inside the while loop

* add **int\_num** to the end of **long\_num**
* get user input for **int\_num** again (inside while loop this time)

#### After the loop exits

* print the value of **long\_num**

# [ ] Create the program, run tests



# Task 3

## Fix the Errors

This loop never runs

### This is a logic error - there is no ErrorMessage, but the code doesn't work

# [ ] review the code, run, fix the Logic error

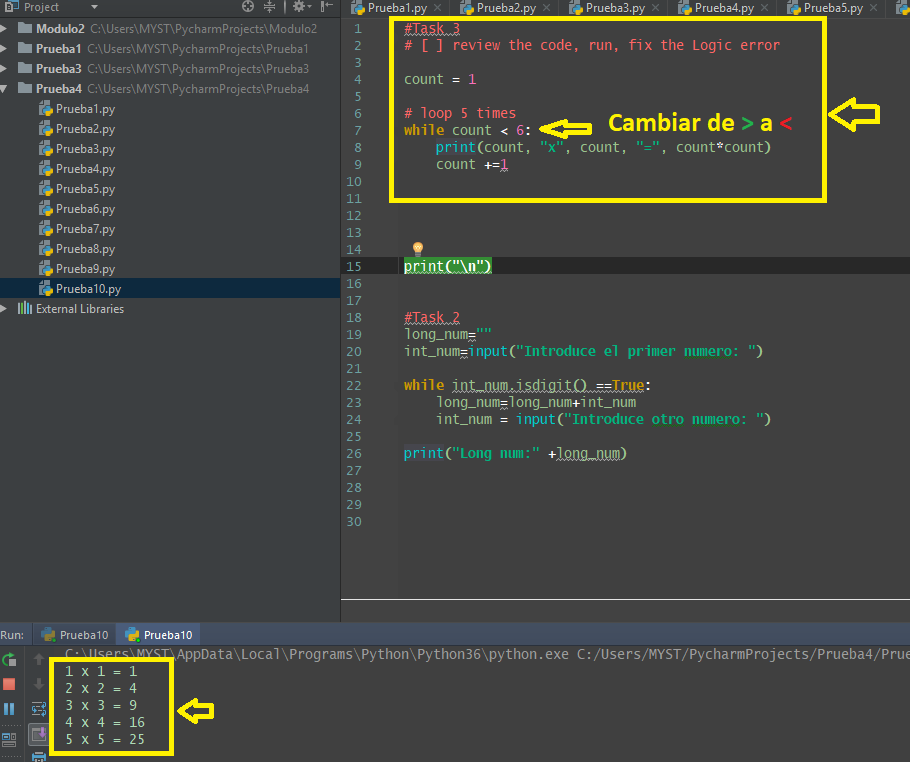
count = 1

# loop 5 times

while count > 6:

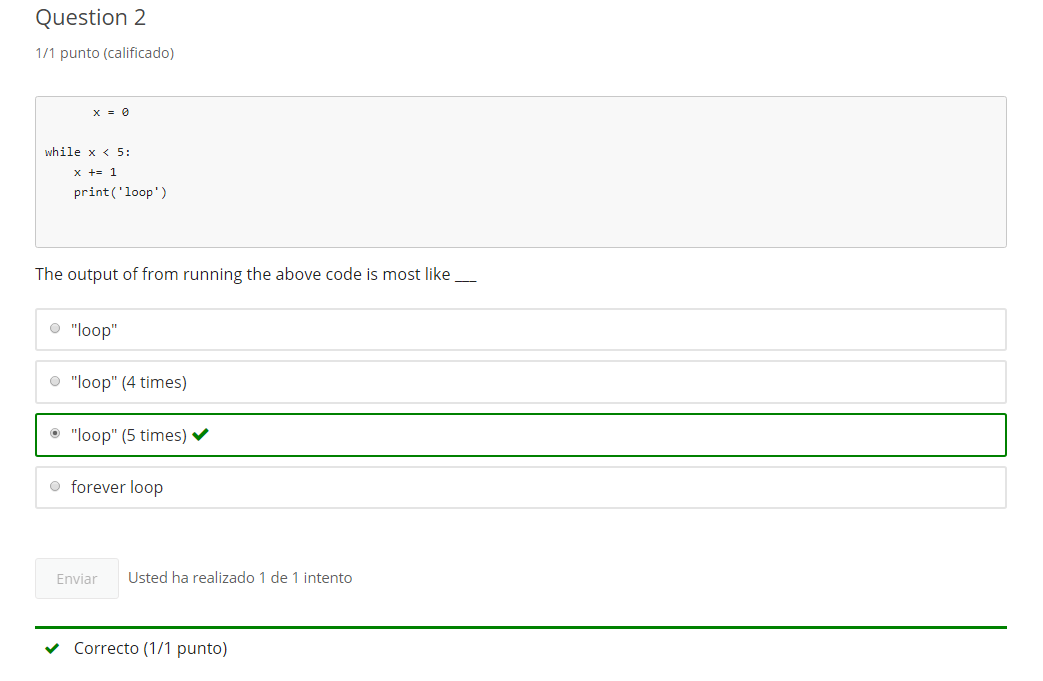
print(count, "x", count, "=", count\*count)

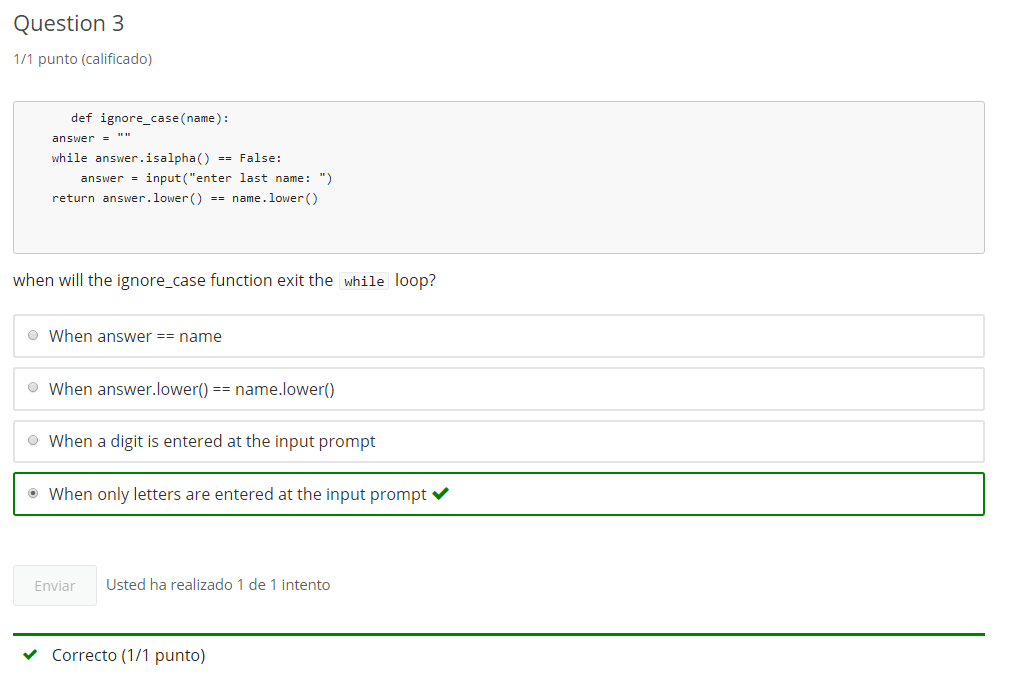
count +=1

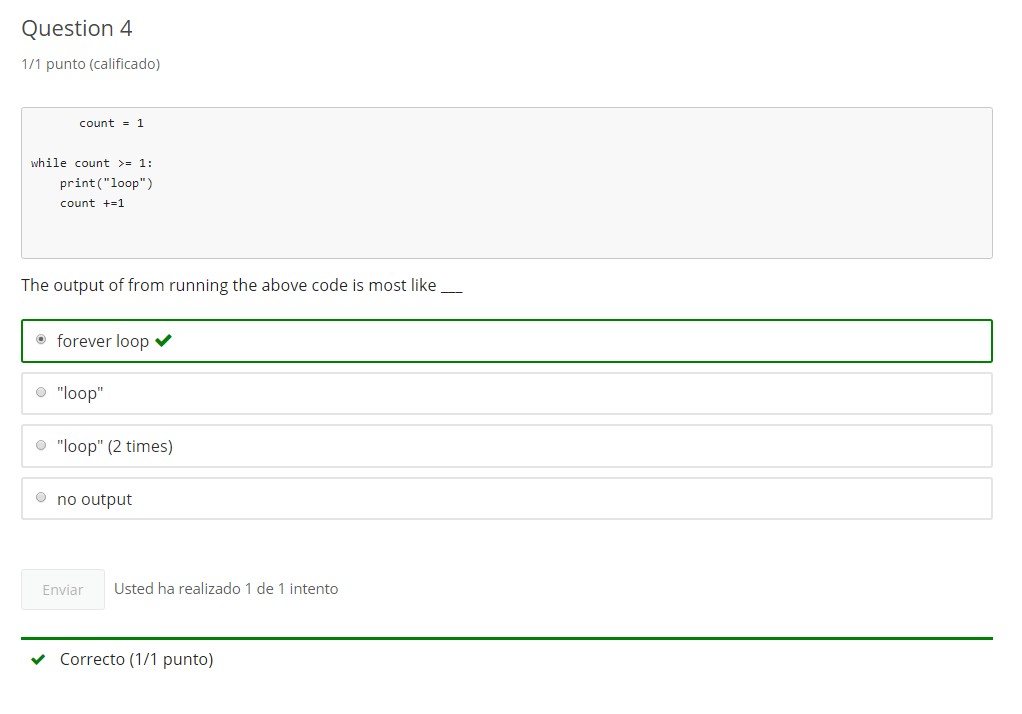


**4.4 Self-Check: Module 4 - Section 4**









**5. Nesting and Loop practice**

**5.1 Module 4 Practice - notebook 6**

# 1-6 Intro Python Practice

optional practice

Jupyter Notebook: MOD04Practice\_1-6\_IntroPy.ipynb

## Nested Conditionals

# Student will be able to

* create nested conditional logic in code
* print format print using escape sequence (\*\*\*\*)

# Tasks

# [ ] print a string that outputs the following exactly: The new line character is "\n"

# [ ] print output that is exactly (with quotes): "That's how we escape!"

# [ ] with only 1 print statement and using No Space Characters, output the text commented below

# 1 one

# 22 two

# 333 three

## Program: quote\_me() Function

quote\_me takes a string argument and returns a string that will display surrounded with **added double quotes** if printed

* check if passed string starts with a double quote ("\""), then surround string with single quotations
* if the passed string starts with single quote, or if doesn't start with a quotation mark, then surround with double quotations

Test the function code passing string input as the argument to quote\_me()

# [ ] create and test quote\_me()

### Program: shirt order

First get input for color and size

* White has sizes L, M
* Blue has sizes M, S

print avaiable or unavailable, then  
print the order confirmation of color and size

\* **hint**: set a variable "available = False" before nested if statements and  
change to True if color and size are avaiable\*

# [ ] create shirt order using nested if

## Program: str\_analysis() Function

Create the str\_analysis() function that takes a string argument. In the body of the function:

* Check if string is digits
  + if digits: convert to int and check if greater than 99
    - if greater than 99 print a message about a "big number"
    - if not greater than 99 print message about small number
  + if not digits: check if string isalpha
    - if isalpha print message about being all alpha
    - if not isalpha print a message about being neither all alpha nor all digit

call the function with a string from user input

# [ ] create and test str\_analysis()

### Program: ticket\_check() - finds out if a seat is avaiable

Call ticket\_check() function with 2 arguments: section and seats requested and return True or False

* **section** is a string and expects: general, floor
* **seats** is an integer and expects: 1 - 10

Check for valid section and seats

* if section is general (or use startswith "g")
  + if seats is 1-10 return True
* if section is floor (or use starts with "f")
  + if seats is 1-4 return True

otherwise return False

# [ ] create and call ticket\_check()

**5.2 Module 4 Practice - notebook 7**

# 1-7 Intro Python Practice

optional practice

Jupyter Notebook: MOD04Practice\_1-7\_IntroPy.ipynb

## while() loops & increments

# Student will be able to

* create forever loops using while and break
* use incrementing variables in a while loop
* control while loops using Boolean operators

# [ ] use a "forever" while loop to get user input of integers to add to sum,

# until a non-digit is entered, then break the loop and print sum

sum = 0

# [ ] use a while True loop (forever loop) to give 4 chances for input of a correct color in a rainbow

# rainbow = "red orange yellow green blue indigo violet"

# [ ] Get input for a book title, keep looping while input is Not in title format (title is every word capitalized)

title = ""

# [ ] create a math quiz question and ask for the solution until the input is correct

### Fix the Error

# [ ] review the code, run, fix the error

tickets = input("enter tickets remaining (0 to quit): ")

while tickets > 0:

# if tickets are multiple of 3 then "winner"

if int(tickets/3) == tickets/3:

print("you win!")

else:

print("sorry, not a winner.")

tickets = int(input("enter tickets remaining (0 to quit): "))

print("Game ended")

### create a function: quiz\_item() that asks a question and tests if input is correct

* quiz\_item()has 2 parameter **strings**: question and solution
* shows question, gets answer input
* returns True if answer == solution or continues to ask question until correct answer is provided
* use a while loop

create 2 or more quiz questions that call quiz\_item()  
**Hint**: provide multiple choice or T/F answers

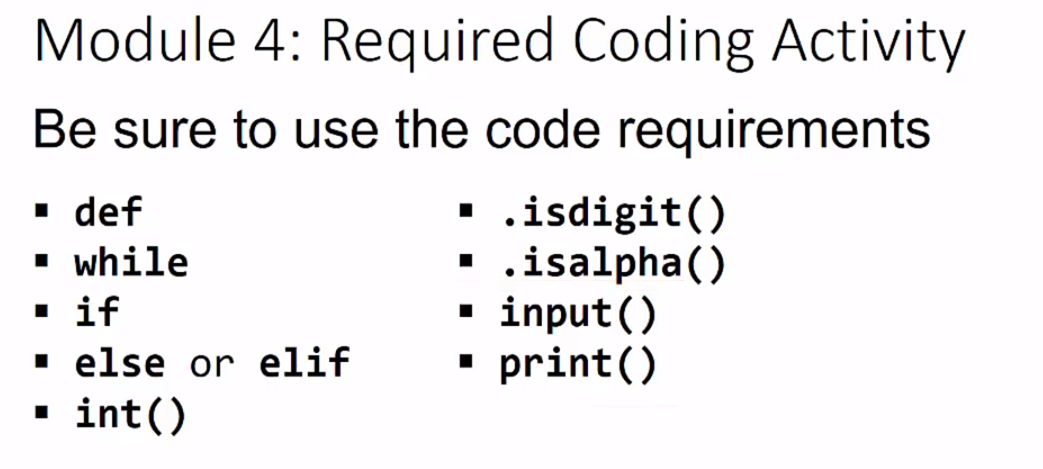
# Create quiz\_item() and 2 or more quiz questions that call quiz\_item()

**6. end of Mod coding assignment**

**6.1 Module 4 Required Code Description**

**Video:** **Module4DesquiredCodeDescriptionV7.mp4**

We're going to review module four: Required Coding Activity on string analysis. And in this activity, you have some key word requirements listed below. We have to define a function and use the def key word. We'll have a while loop, that will loop while we're inputting a valid input, so we can't have an empty string entered. We're going to use if, and then we're going to either use else or elif. Also, we'll be using the int, to cast input into a integer or a digit. We're going to check if input is digit or is alpha. And of course, we'll have to get user input and we will be printing some results to our users.



Let's look at the edx page for this coding activity. And like the other pages, there are the assignment requirements, that we just looked at. And we'll have that same information on our Jupiter notebook. We should use the Jupiter notebook to code our solution. I've pre-loaded this page with a solution, so that we can see how it runs. So it asks us to enter a word or an integer. If we enter nothing, it will continue to ask us for input. So, let's go ahead and just run that again. Okay, so now we can enter in a word, so which is right word. And we see that word is all alphabetical characters. So it checks if the input is all alpha. For that message, we can run it again and I can put in a series of digits. And it says, "that's a pretty big number", because that's bigger than 99. So let's go ahead and run it again. I could run it with something smaller. And it says that is "28 is smaller than expected". We can also just see how the behavior works otherwise. So if I say 22.5, it says "that's not all alpha or all digit characters". And some of us might disagree with that, but it's looking at the characters, it's not looking at the data type. And this may look like it is a float, but is actually a string, and it's a string that contains a non-digit. So likewise, if we are running, it asks us for a word or integer, so if I put Alton Derosoa, it's also not an alpha or a digit because a space is not an alphabetical character or a digit. So, you can go ahead and build that code. Make sure you keep getting input until it has something in there that it checks for the empty input. And then we test whether it is an integer or digit. If it's a digit, we're going to have to convert that into a number. See how big it is, and use some nested comparisons. So, make sure we're going to use all of these keywords. And if you're having trouble submitting, look at some of the earlier required coding activity videos.

# Module 4 Required Coding Activity

Introduction to Python Unit 1

**This Activity is intended to be completed in the jupyter notebook, Required\_Code\_MOD4\_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.

The activity is based on modules 1 - 4 and is similar to the Jupyter Notebooks **Practice\_MOD04\_1-6\_IntroPy.ipynb** and **Practice\_MOD04\_1-7\_IntroPy.ipynb** which you may have completed as practice. This activity is a new version of the str\_analysis() function.

| **Assignment Requirements** |
| --- |
| This program requires the use of **while** loop to get non-empty input, **if, else**, **if, else** (nested), **.isdigit()** check for integer only input, **.isalpha()** check for alphabetic only input. The program should **only** use code syntax covered in modules 1 - 4. The program must result in printed message analysis of the input. |

## Program: str\_analysis() Function

Create the str\_analysis() function that takes 1 string argument and returns a string message. The message will be an analysis of a test string that is passed as an argument to str\_analysis(). The function should respond with messages such as:

* "big number"
* "small number"
* "all alphabetic"
* "multiple character types"

The program will call str\_analysis() with a string argument from input collected within a while loop. The while loop will test if input is empty (an empty string "") and continue to loop and gather input until the user submits at least 1 character (input cannot be empty).

The program then calls the str\_analysis() function and prints the **return** message.

#### Sample input and output:

enter nothing (twice) then enter a word

enter word or integer:

enter word or integer:

enter word or integer: Hello

"Hello" is all alphabetical characters!

alphabetical word input

enter word or integer: carbonization

"carbonization" is all alphabetical characters!

numeric inputs

enter word or integer: 30

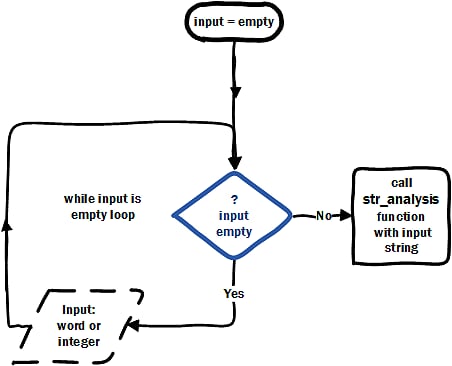
30 is a smaller number than expected

enter word or integer: 1024

1024 is a pretty big number

### loop until non-empty input is submitted

This diagram represents the input part of the assignment - it is the loop to keep prompting the user for input until they submit some input (non-empty).



Once the user gives input with characters use the input in calling the str\_analysis() function.

### Additional Details

In the body of the str\_analysis() function:

* Check if string is digits
  + if digits: convert to int and check if greater than 99
    - if greater than 99 print a message about a "big number"
    - if not greater than 99 print message about "small number"
  + check if string isalpha then (since not digits)
    - if isalpha print message about being all alpha
  + if not isalpha print a message about being neither all alpha nor all digit

call the function with a string from user input

* Run and test your code before submitting

# [ ] create, call and test the str\_analysis() 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/)

**2.2 Module 4 Required Code Submission**

