Hi, welcome to Basic Program Structure.

☼ Repetition , that is, Looping in Python. We will discuss how to execute a set of instructions repeatedly, most importantly, to control the condition that allows us to keep repeating those instructions.

☼After this lesson, you should be able to:

- Write programs for executing statements repeatedly using a ***while*** loop in Python
- Describe how to control a loop with sentinel value in Python
- Write loops using ***for*** statements in Python
- Discuss how to generate a sequence of numbers using the ***range()*** function
- Apply ***for*** loop to iterate through a ***range*** sequence
- Write nested loops in Python
- Implement program control with ***break***, ***pass***, and ***continue*** in Python

☼

In this lesson, we will cover a few topics. Let's go through them in details one by one.

☼Python offers two different styles of repetition,: while and for.

While loops are Usually used for sentinel-controlled loops, but may also be used to implement counter-controlled loops.

For loops are usually used for counter-controlled loops.

We will introduce while loop first.

☼

The while statement introduces the concept of repetition The while statement allows repetition of a suite of Python code as long as some condition (Boolean expression) is True.

A while statement is structurally similar to an if statement. It consists of a header (which has the loop condition) and a suite of statements.

The test condition is a Boolean expression. We evaluate the condition and, if the condition is True, we execute all of the while suite code. At the end of that suite, we again evaluate the condition. If

it is again True, we execute the entire suite again. The process repeats until the condition evaluates to False.

When the condition becomes False, repetition ends and control moves on to the code following the repetition.

☼

The while loop contains a Boolean decision that can be expressed in English as:

"While the Boolean expression is True, keep looping—executing the suite."

Let's take a look at a simple example that prints out the numbers zero through nine.

☼Before the loop begins, we initialize the variable x to be 0. We call this our loop-control variable, as its value will determine whether we continue to loop or not.

The while loop works as follows: The program enters the while construct and evaluates the Boolean expression(while x < 10). Because we initialized x to be zero, this Boolean expression will be True for x values of 0 to 9 but will be False for larger integer values.

0<10, is true, then the associated while suite is executed, 0 is printed.

In the final instruction of the suite, we change the value associated with the

loop-control variable. In this case, we increase its present value by 1 to be 1 and reassign it to the variable.

At the end of the suite, control flows back to the top of the while, where the Boolean expression is reevaluated. 1<10, is true, then the associated while suite is executed, 1 is printed.

Loop-control variable increase by 1 to be 2. control flows back to the top of the while, where the Boolean expression is reevaluated, the process repeated until x becomes 10. Over the course of running the while loop, x will take on the integer values of 0–10 but note that 10 is not printed in the loop.

10<10, is false. When the Boolean expression yields False, the while suite is skipped and the code following the while loop is executed. "Final value of x int: 10." is displayed.

Some things to note about how a while statement works. The condition is evaluated before the suite is executed. This means that if the condition starts out False, the loop will never run.

Further, if the condition always remained True, the loop will never end. Imagine if we did not add 1 to x each time above, the condition would always be True (x started with a value of 0, which is less than 10, and never changes through each loop iteration) and the program loops forever.

Notice the output of our example? One number per line. Can we display them in the same line?

☼

Yes, by adding one more parameter to the built-in function print().

The end equals to empty string in the print statement indicates that the print ends with an empty string rather than the default new line. This means that the output from multiple calls to print will occur on the same output line.

We can likewise end with any symbol. For example "@". the output is number 0 to 9 separated by @.

☼

Quick check: **What is the output of the following code?**

☼Answer:

This question is very similar to our example. The only difference is that there is only one single statement in while suite, that is, count = count + 1

The print statement is the code following the while loop. It executes only once when the loop condition becomes false. Count is initialized to be 5. The condition is count<9. it is only when the count is equal to 9 that the condition is false. So, print count = 9.

☼

Let's discuss a bit more complex question: looping and selection, with arithmetic operators.

☼

Answer is

A. count is: 3 number is: 0

why? The count and number are initialized to be 0 and 9 respectively. Loop-continuation condition is, number > 0. The current number is 9, which is greater than 0. Thus, the condition is true. Enter the while loop. 9 is an odd number, %2 == 0 is false, so it goes to elif. 9%3 == 0 is true. 9 floor division 3 is 3. Skip else statement, increase count by one to be 1.

The control flows back to the top of the while, where the Boolean expression is reevaluated. current number is 3, which is greater than 0. So the condition is true. Enter the while loop. 3 is an odd number, %2 == 0 is false. So it goes to elif. 3%3 == 0 is true. 3 floor division 3 is 1. Skip else statement, increase count by one to be 2.

The control flows back to the top of the while, where the Boolean expression is reevaluated. The current number is 1, which is greater than 0. The condition is true. Enter the while loop. 1 is an odd number, %2 == 0 is false. So it goes to elif. 1%3 == 0 is false. Then execute else statement, decrease number by 1. The current number becomes 0. Increase count by one to be 3.

The control flows back to the top of the while, where the Boolean expression is reevaluated. The current number is 0, which is not greater than 0. Skip the while suite and it goes to the code following the while loop. Display the current count, which is 3 and current number, which is 0.

☼Now let's continue our analysis,

If line 1 and line 2 were removed, what effect would that have on the program?

☼

The answer is "the while loop would not end". There is no control variable update when the number is neither an even number nor divisible by 3, hence, the condition remains true. The loop will never end.

☼Let's start to use looping to solve some problem in real life. Recall our scenario of finding the distances to N locations.

We already have the flowchart and pseudocode. How to convert them into a Python program that runs on a computer?

☼**Let's follow the three steps in Loop Design Strategies**

Step1: Identify the statements that need to be repeated.

We already have the code from previous lesson.

horizonDist = int(input("Read horizonDist in meters"))vertDist = int(input("Read vertDist in meters"))dist = horizonDist + vertDistprint("dist from home to coffee shop ", count, " is ", dist, "m")

☼Step2: Wrap these statements in a python while loop

While true:

```
    horizonDist = int(input("Read horizonDist in meters"))
    vertDist = int(input("Read vertDist in meters"))
    dist = horizonDist + vertDist
     print("dist from home to coffee shop ", count, " is ", dist, "m")
```

Pay attention to the colon and indentation

☼

Step 3: Code the loop-continuation-condition and add appropriate statements for controlling the loop.

Decide the loop-continuation-condition to be when count <= N is true, enter the loop body, and Code it using relational operator <= , count <= N

Initialize the loop control variable count

Update the loop control variable count

It's a good programming style to add User-friendly message indicating index of current input/loop control variable when necessary.

☼

Let's Check loop iteration when N = 2.
Let's think about the following questions:

First, in the loop, what is the loop control variable? Count

Second, Is it a counter-controlled or sentinel-controlled? Counter-controlled. The loop body is repeated for two times, and the number of repetitions is known before the loop begins execution.

Then we can use a table to check.

The loop control variable count is initialized to be 1.

Check loop-continuation-condition 1<=2, which is true. Enter the loop body. Display distance from home to first coffee shop

The loop control variable count is updated to be 2.

Check loop-continuation-condition 2<=2, which is true. Enter the loop body. Display distance from home to second coffee shop

The loop control variable count is updated to be 3.

Check loop-continuation-condition 3<=2, which is false, The loop is terminated. The program flow will continue with the first statement after the while statement, which is to display the Thank you message.

☼

This slide shows the outputs of the sample runs based on different values of N.

So ,the program instructions can be repeated dynamically without changing the code. Two times, One time, Or does not even repeat at all.

☼From the last sample run, have you found any potential issue in this programme

What will happen if the user enters a value smaller than 1? Or 99999999999?

-2 coffee shop, quite odd. 99999999999, really too many, So… Any idea to fix it?

☼Maybe… we can force the user to input again if the input is out of reasonable range? But how?

Maybe we can Keep asking until he/ she enters a number that is at least 1 and at most a certain reasonable limit?

How to code? We can add a while loop.

☼

a new while loop is added.

( ) round bracket is ok if you are used to it in other programming language but redundant in Python

In the new while loop: What is the loop control variable? N

Is it a counter-controlled or sentinel-controlled Loop? sentinel-controlled. We cannot control user's input, so the number of repetitions is not known before the loop begins execution.

Note while loop is useful if we want to repeatedly ask user for input until the input fulfills our requirement.

☼One more common sentinel-controlled loop example. Write a program to read numbers from a user; Sum them up until the input is -1. The output of a sample run is shown here. The program does not know how much input is to be read at the beginning of the program. However, it will keep on reading the data until the user inputs -1, which is the sentinel value.

☼

Common Sentinel Loop Implementation is in the form illustrated with sample code in this slide.

(value = some value) sample code: (item = float(input("Enter value:"))

get initial value of loop control variable

while value != sentinel value: sample code: (while item != -1:

) if the value is not equal to the Sentinel value

(*process value*) sample code: (total += item) execute loop body

(*get another value*) sample code: (item = float(input("Enter value:")) to update the loop control variable

Any issue to ensure for this design? What will happen if the program is to calculate the balance of a bank account? There is a chance for -1 to be a normal input. The program will be terminated once it receives a -1. All the numbers after -1, cannot be entered.

So, selection of sentinel value is extremely important; programmers should make sure that Sentinel value will not appear in normal inputs.

☼

Similar to the if-else statement, you can have an else clause at the end of a while

loop. else statement is executed when the loop finishes under normal conditions It is the last thing the loop does as it exits

The else is strictly optional, but it does have some very nice uses.

☼

The else clause is entered after the while loop's Boolean expression becomes False. For example, after two rounds of true paths, it enters false path, execute else suite.

This entry occurs even if the expression is initially False and the while loop never ran (its suite was never executed).

As with other header statements, the else part of the while loop can have its own associated suite of statements.

Think of the else statement on the while loop as "cleanup" for the action performed as the loop ends normally.

The else clause is a handy way to perform some final task when the loop ends normally.

☼

Here is the modified version of reading in number program.

The statements in the else part are executed, when the condition is not fulfilled anymore. It doesn't matter the while loop has been run or never ran.

☼quick check:What is the output of the following code?

☼The answer is:

```
before while 1
while 2
while 3
while 4
else 4
```

There is nothing special, the only new knowledge point is that the else clause is entered after value becomes 4 and the while loop's Boolean expression becomes 4<=3 which is False, so else 4 is printed.

At this point some might ask themselves , where the possible benefit of this extra branch is. If the statements of the additional else part were placed right after the while loop without an else, they would have been executed anyway, wouldn't they?

☼We need to understand a new language construct, i.e. the break statement, to obtain a benefit from the additional else branch of the while loop.

The **break** statement can be used to *immediately* exit the execution of the *current* loop and skip past all the remaining parts of the loop suite.

It is important to note that "skip past" means to skip the else suite (if it exists) as well.

The break statement is useful for stopping computation when the "answer" has been found or when continuing the computation is otherwise useless.

☼Let's understand it better through a simple example.

Quick try:

Compared to quick check question: We add if and break

Knowledge point: Executing break exits the immediate loop that contains it. It goes after the whole enclosing loop.

In this program, the program flow will continue with the first statement after the while statement, which is to display "after while" and value.

☼The answer is:

```
before while 1
while 2
after while 2
```

And no more

while3
while 4
else 4

Why? When value becomes 2, execute break. Executing break exits the immediate loop that contains it, and continue the first statement after the while statement.

☼We like the clarity/ˈklarɪti/ of the while and if statements. They make it clear to the reader how the program might enter and exit the various suites and, as we have said repeatedly, readability is important for correctness. However, sometimes we need to take a non-normal exit, and that is the purpose of a break.

Is executing a break statement a non-normal exit? We choose a non-normal exit sparingly because it can detract from readability. But sometimes it is the "lesser of the two evils" and it provides the best readability and, on occasion, a better performance. A non-normal exit is sometimes called an early exit.

☼To illustrate while-else, a short program that plays a "hi-low" number guessing game is given here. The program starts by generating a random number hidden from the user that is between 0 and 100. Since we have not learned the random module yet, for simplicity, we hardcode the number. The user then attempts to guess the number, getting hints as to which direction (bigger or smaller, higher or lower) to go in the next guess. The game can end in one of two ways:

The user correctly guesses the number.

The user can quit playing by entering a number out of the range of 0–100.

Here is an algorithm for the game:

Choose a random number.

Prompt for a guess.

while guess is in range:

    Check whether the guess is correct;

        If it is a win, print a win message and exit.

        Otherwise, provide a hint and prompt for a new guess.

Else:

  user quit

☼ The source code is given.

Let's analyze the code.

You are familiar with the rest parts of the program. So, let's focus on the while statement.

This is the "normal" exit condition for the loop. As long as the guess is in the

range 0–100 inclusive, the loop will continue. As soon as the expression becomes False, the loop will end and the associated else suite will be run. You get to this print statement "a pity.." only if the while loop terminated normally.

so that you know without further checking that 0 <= guess <= 100 is no longer

True and the user quits.

For Loop body: If the guess is incorrect (either too high or too low), we provide a hint to the user so that he or she could improve the next guess.

If it wasn't high and it wasn't low, it must be the answer. Is that true—or

could it have been a number not in the range we were looking for (0–100)? No, we checked that in the while Boolean expression. It must be the correct number. We print a winner message and then use the break statement to exit the loop.

The else clause is often used in conjunction with the break statement.

☼Remember, a break will skip the else suite of the while loop!

☼Sometimes we might want to simply skip some portion of the while suite we are executing and have the control flow back to the beginning of the while loop. That is, exit early from this iteration of the loop (not the loop itself), and keep executing the while loop. In this way, the continue statement is less drastic than the break. The continue statement, continues to the next iteration of the loop.Similar to the break, it can make the flow of the control harder to follow, but there are times when it yields the more readable code.

☼Consider writing a program that continuously prompts a user for a series of even integers that the program will sum together. If the user makes an error and enters a non-even (odd) number, the program should indicate an error, ignore that input, and continue with the process. Given this process, we need a way to end the loop. Let's choose a special character to stop the looping. In our case, if the special character "." is entered, the program will print the final sum and end.

☼Here's the detailed implementation. First Initialize the variables. Number_str is set to the result of a input request to the user.

We append "str" to the variable name so that we may keep straight that.in fact, it is a string, not a number. the sum is initially set to 0. We use the name the_sum because the name sum is a built-in function name that will be explained later.

The period (.) or full stop is our special indicator that the loop should end. so we

loop until a user inputs a period or full stop. If the input is not a period or full stop. Enter loop body.

Convert the input string to a number (int).

Here we check whether the number is even or odd by finding the remainder

when it is divided /dɪˈvaɪdɪd/ by 2 (using the % operator).

If it is odd, we print the error message

and then re-prompt /prɒmpt/ the user. We continue which means that we go to the

top of the while loop and evaluate its Boolean expression, skipping over the rest of the suite.

Check the loop continuation condition if the input is not a period. Enter loop body.

Convert the input string to a number (int).

Check whether number is even or odd.

If it is even, include it in the sum. We then prompt /prɒmpt/ the user for a new number.

It repeats until a period is entered.

After the loop ends, print the sum of the integers accumulated in the variable

the sum.

Control flow knowledge: continue statement. It goes immediately to test

in the enclosing loop.

Note that For **continue**, make sure that the loop control variable can be updated. Or Else it becomes **infinite loop**!

☼Python offers two different styles of repetition: while and for. The while statement introduces the concept of repetition. The while statement allows us to repeat a suite of Python code as long as some condition (Boolean expression) is True.

The for statement, on the other hands,implements iteration. Iteration is the process of examining all elements of a collection, one at a time, allowing us to perform some operations on each element.

It has the ability to iterate over the items of any sequence, such as a list or a string.

Each item in the sequence is assigned to the iterating variable iterating_var. Next, the statements block is executed.

The for loop completes when the last of the elements have been assigned to the iterating_var. i.e., the entire sequence is exhausted /ɪgˈzɔːstɪd/.

☼

Like the if and while statements, the for statement has a header and an associated suite.

Two Keywords: for in are used.

The keyword in precedes the sequence.

The variable iterating_var is a variable associated with the for loop that is assigned the value of an element in the sequence.

The variable iterating_var is assigned a different element during each pass of the for loop.

Eventually, iterating_var will have been assigned to each element in the sequence.

☼

Let's look at the example with two for structures and analyze the first one in details.

Because the collection in this case is a string, the variable the_char is assigned, one at a time, to each character in the string. Since a string is also a sequence (i.e., an ordered collection) the variable the_char is assigned in order from the first character, "t", to the last, "g".

We will talk much more about strings in later lessons. As with a while loop, the entire for suite is evaluated during each pass of the loop. One can therefore perform operations on each element of a collection in a very straightforward way. In this particular case, we print only the value of the_char each time the for loop iterates. The for loop completes when the last of the elements has been assigned to the char. The result in this case will be a printout with eight lines, each line printing out one character.

Second example is a list as the sequence. We will discuss list in details in later lessons. Here, we only need know that Python allows us to examine each individual element in the sequences, one at a time.

☼

Quick check:

- write a python program to find the sum of the numbers from 1 to 10 using for loop.

The answer is : sum = 0

```
for i in [1,2,3,4,5,6,7,8,9,10]:
  sum += i
print ("the sum of the numbers from 1 to 10 is ", sum)
```

The for loop has the loop control variable (in the case above it is i) take on each of the values in the list, in left to right order, and then executes the connected suite, that is, the loop body executes once for each value of **i** to add onto the previous sum. The loop will therefore execute the same the number of times as the number of elements in the list.

Let's use the output to verify. There are 10 elements in this list, from the output, the loop body was executed 10 times.

☼

Sometimes it may be necessary to have the loop execute a great many times.

If the loop were to execute a million times, it would be more than awkward to

require a program to list a million integers in a sequence. Python provides a function to make this more convenient: range(). It returns a sequence that consists of all of the integers between the two arguments it is given, including the lower endpoint.

It takes up to three arguments: the start value, if omitted , the default value is 0, the end value, which is mandatory, cannot be omitted , and the step or separation between each value, if omitted , the default value is 1. The range function generates what is termed a half-open range. Such a range does not include the end value in the sequence generated.

Please note that All parameters must be integers, positive or negative

☼range() function has three forms with different number of parameters:1, 2, 3 respectively.

A form with one parameter only explicitly specifies end value, start and step use default values, which are 0 and 1 respectively.Here is an example: range (11), which is equivalent to range(0, 11,1). It generates a sequence of integers which starts from default value, 0, ends at 10, that is, 11-1, and with step of default value 1. So it gives, 0, 0+1=1, 1+1 =2, 2+1=3, 3+1=4, and so on, until 9+1=10 and stops. Range(11) gives [0,1,2,3,4,5,6,7,8,9,10]

A form with two parameters, the start and end uses a default step, which is 1 For example, range (1,11) which is equivalent to range(1, 11,1), generates a sequence of integers that starts from 1, ends at 10, that is 11-1, with a step of default value 1. So it gives, [1,2,3,4,5,6,7,8,9,10]

Here are exampes of range functions with three parametes: range (1,11,2). It generates a sequence of integers that starts from 1, ends at 10, that is 11-1, with a step of 2. So it gives, 1, 1+2=3, 3+2=5, 5+2 =7, 7+2=9, 9+2=11 exceed 10, and stops Range(1,11,2) gives [1, 3, 5, 7, 9]

range(11, 2, -3) gives [11, 8, 5], You can step backwards easily by setting the step argument to be negative.

☼quick check. What is the output of the following code?

☼The answer is 45. The program is to find the sum of the numbers from 1 to 9. Why 9 instead of 10? The range function in Python excludes the ending element.

☼One more quick check question. What is the output of the following code? In this program, there are two for loops with same loop control variable.

☼ The answer is shown in the slide. In the first for loop, add the numbers 1,2,3, and 4 together. We get 10In the scond loop, it starts from 5, ends at 1 with a step size of -1, that is step backwards. So ,the sequence generated is 5,4,3,2, add them together, we get 14.

SO take note:first, the sequence is not symmetric , range (1,5,1) != range (5,1,-1). Second, variable number in the example "for" loops will take different values in different loop iterations. For every loop iteration, it works as a new variable.

☼Let's apply our knowledge to work on a very common problem. Ask the computer to Print a multiplication table of a certain number (multiplier). For example, if the number is 9. The output will be like this. We can imagine the logic of the program. First, ask the user for a number. Then, loop from 1 to 10, compute the multiplication, and display the formulas.

☼How to implement it in Python? In Python, we just use a few lines…the suggested code is given. Note that we end with 11 instead of 10 if we want the last value to be 10. Use comma to separate different elements for print.

☼The for statement, just like the while statement, can support a terminal else suite, as well as the control modifiers, continue and break. Their roles are as previously described, which are:

1. The else suite is executed after the for loop exits normally.

2. The break statement provides an immediate exit of the for loop, skipping the else clause.

3. The continue statement immediately halts /hɔːlt/ the present iteration of the loop, continuing with the rest of the iterations.

☼quick check. What is the output of the following code?

☼Answer is Pyt. When the for loop comes to letter h the condition is true .Execute break statement, and exit the for loop immediately

☼quick check. What is the output of the following code?

☼Answer is Pyton, just screen letter 'h'. When the for loop goes to letter h, the condition is true. Execute continue statement, halts the present iteration of the loop, and continues with the rest of the iterations.

☼ Here's a program that can illustrate the use of a for-else loop, and some other ideas as well. Determine whether a number is prime or non-prime. A prime number does not have other divisors except 1 and itself; So, to find out whether a number is prime, try dividing it by numbers smaller than it, and if any of them have a zero remainder then the number is not prime. This is a job for a for loop. Here's the simplest logic:

isprime = True

for n in range (1, K):

if k%n == 0:

isprime = False

After the loop is completed, the variable 'isprime' indicates whether K is prime or not. This seems pretty simple, but tedious. It does a lot of divisions. Too many, in fact, Is there a need to try dividing it by all numbers smaller than it? No. because it is not possible for any number larger than K/2 to divide evenly into K.

☼The suggested code is given. The idea that the loop can be exited explicitly makes the normal termination of the loop something that should be detectable too. When a while or for loop exits normally by exhausting the iterations or having the expression become False, it is said to have fallen through. When the for loop in the prime number program detects a factor, it now executes a break statement, thus exiting the loop.

The else part of a while or for loop is executed only if the loop falls through; that is, when it is not exited through a break. This can be quite useful, especially when the loop is involved in a search, which will be discussed later. In the case of the prime number program, an else could be used when the number is in fact prime

☼ Just as it is possible to have if statements nested within other if statements, a loop may appear inside another loop. This is called Nested Loop.

That is an outer loop may enclose an inner loop.

We can nest as many levels of loops as the system allows.

We can also nest different types of loops.

☼The program contains a nest loop, in which one ‘for loo’p is nested inside another for loop. The counter valiable y and x are used to control the for loops. This is a common application of two-levels nested ‘for loop’ to display two dimensional array.

This is the entire suite/ block inside the outer ‘for loop.’

There is Only one statement inside the inner ‘for loop’.

End = ‘’empty string, This additional argument is used in Python 3 to avoid the default ending return \n

- How many times is each print executed? What is the control flow?

The nested loop is executed in this way. In the outer ‘for loop’, when the counter variable y = 1, the inner loop is executed for 5 iterations, and generates (1 , 1) (2 , 1) (3 , 1) (4 , 1) (5 , 1). Then executes outer print to change to a new line, when y =2 in the outer loop, the inner loop is executed for 5 iterations, and generates (1 , 2) (2 , 2) (3 , 2) (4 , 2) (5 , 2), then executes outer print to change to a new line. Similarlly, when y =3 in the outer for loop, the inner for loop is executed for 5 iterations and generates (1 , 3) (2 , 3) (3 , 3) (4 , 3) (5 , 3) then executes outer print to change to a new line, The nested loop is then terminated.

So, the inner print in this example would execute 15 times. Each time the outer loop executes the inner one is executed for 5 times, for a total of 3*5 or 15 iterations.

The outer print in this example would execute 3 times.

☼ We have Printed a multiplication table of a certain number. How to Print a full Multiplication Table? Add one more layer of for loop. The suggested code is given here and the output of sample run.

☼Loops can be nested to a greater depth if necessary, and while and for loops can be nested interchangeably. Is it ever useful to do this? Yes, very often.

Let’s extend out previous example of test a number is prime or not to Test multiple numbers are prime or not?

How many numbers to check? Unknown. User enter '#' to end the checking

It is clear that this is a sentinel control loop, which is a while loop

☼We already have code to test a single number is prime or not. So let's build on it.

While loop is added to read the user's input repeatedly.

Note: break affects only the inner loop that contains it.

Here an output of a sample run.

☼A pass statement has no effects (do nothing), but helps in **indicating** an empty statement/ suite/ block

You use pass when you have to put something in a statement (syntactically, you cannot leave it blank or Python will complain) but what you really want is nothing.

Python has the syntactical requirement that code blocks (after if, for, while, except, def, class etc.) cannot be empty.

For example, python will complain if you put nothing after for, if you put pass after it, the for statement will iterate through the range values but do nothing with them.

pass has its uses. It can be used to test a statement (say, opening a file or iterating through a collection) just to see if it works. You don't want to do anything as a result, just see if the statement is correct.

More interestingly, you can use pass as a place holder. You need to do something at this point, but you don't know what it is yet. You place a pass at that points and goes back later to fill in the details.

☼quick check:

What is the output of the following code?

☼The answer is python.

Pass does nothing, so there is nothing special that would happen when the letter is h.

☼Quick summary:

The ability to repeat a collection of operations is an essential part of any programming language.

The while loop has a condition at the beginning, and as long as that condition is true, the statements comprising the loop will be executed repeatedly.

The for loop has an explicit list of items for which the loop will be

executed, or a range of numerical values that defines how many times the code

will be repeated.

The range is an important function in python

The start value is the first value included in the sequence and, if not provided, defaults to 0. The end value is used to determine the final value of the sequence. Again, the end value itself is never included in the sequence. The end value is a required argument. The step value is the difference between each element generated in the sequence and, if not provided, defaults to 1. If only one argument is provided, it is used as the end value. In the case of one argument, constituting the end value, the start value is assumed to be 0 and the step value assumed to be 1. If two arguments are provided, the first is the start and the second is the end. There is a step argument only if three arguments are used.

Keywords in loop structure: else, break, continue, pass 1. The else suite is executed after the for loop exits normally. 2. The break statement provides an immediate exit of the for loop, skipping the else clause. 3. The continue statement immediately halts the present iteration of the loop, continuing with the rest of the iterations. 4. pass has no effects (do nothing), but help indicates an empty statement/ suite/ block

Just as it is possible to have if statements nested within other if statements, a loop may appear inside another loop. The placing of one loop inside the body of another loop is called nesting. When you "nest" two loops, the outer loop takes control of the number of complete repetitions of the inner loop. While all types of loops may be nested, the most commonly nested loops are for loops.

I hope you enjoy this lecture. And, I will see you next time.