Want to try this out? Let's give it a go!

Fill in the gaps of the sum\_squares function, so that it returns the sum of all the squares of numbers between 0 and x (not included). Remember that you can use the range(x) function to generate a sequence of numbers from 0 to x (not included).

def square(n):

    return n\*n

def sum\_squares(x):

    sum = 0

    for n in range(x):

        sum += square(n)

    return sum

print(sum\_squares(10)) # Should be 285

RunReset

Here is your output:

285

Nice job! You've got Python to do some complex operations

for you!

For Loops Recap

*For* loops allow you to iterate over a sequence of values. Let's take the example from the beginning of the video:

for x in range(5):

  print(x)

Similar to *if* statements and *while* loops, *for* loops begin with the keyword ***for*** with a colon at the end of the line. Just like in function definitions, *while*loops and *if* statements, the body of the *for* loop begins on the next line and is indented to the right. But what about the stuff in between the *for* keyword and the colon? In our example, we’re using the *range()* function to create a sequence of numbers that our *for* loop can iterate over. In this case, our variable **x** points to the current element in the sequence as the *for* loop iterates over the sequence of numbers. Keep in mind that in Python and many programming languages, a range of numbers will start at 0, and the list of numbers generated will be one less than the provided value. So *range(5)* will generate a sequence of numbers from 0 to 4, for a total of 5 numbers.

Bringing this all together, the range(5) function will create a sequence of numbers from 0 to 4. Our *for* loop will iterate over this sequence of numbers, one at a time, making the numbers accessible via the variable **x** and the code within our loop body will execute for each iteration through the sequence. So for the first loop, **x** will contain 0, the next loop, 1, and so on until it reaches 4. Once the end of the sequence comes up, the loop will exit and the code will continue.

The power of *for* loops comes from the fact that it can iterate over a sequence of any kind of data, not just a range of numbers. You can use *for* loops to iterate over a list of strings, such as usernames or lines in a file.

Not sure whether to use a *for* loop or a *while* loop? Remember that a *while*loop is great for performing an action over and over until a condition has changed. A *for* loop works well when you want to iterate over a sequence of elements.

In math, the factorial of a number is defined as the product of an integer and all the integers below it. For example, the factorial of four (4!) is equal to 1\*2\*3\*4=24. Fill in the blanks to make the factorial function return the right number.

def factorial(n):

    result = 1

    for i in range(1,n+1):

        result=result\*i

    return result

print(factorial(4)) # should return 24

print(factorial(5)) # should return 120

Here is your output:

24

120

Well done, you! The pieces of code you're tackling keep

getting more complex, but you're doing a great job

## A Closer Look at the Range() Function

Previously we had used the range() function by passing it a single parameter, and it generated a sequence of numbers from 0 to one less than we specified. But the range() function can do much more than that. We can pass in two parameters: the first specifying our starting point, the second specifying the end point. Don't forget that the sequence generated won't contain the last element; it will stop one before the parameter specified.

The range() function can take a third parameter, too. This third parameter lets you  alter the size of each step. So instead of creating a sequence of numbers incremented by 1, you can generate a sequence of numbers that are incremented by 5.

To quickly recap the range() function when passing one, two, or three parameters:

* One parameter will create a sequence, one-by-one, from zero to one less than the parameter.
* Two parameters will create a sequence, one-by-one, from the first parameter to one less than the second parameter.
* Three parameters will create a sequence starting with the first parameter and stopping before the second parameter, but this time increasing each step by the third parameter.
* Given the following code:
* teams = [ 'Dragons', 'Wolves', 'Pandas', 'Unicorns']
* for home\_team in teams:
* for away\_team in teams:
* What should the next line be to avoid both variables being printed with the same value?
* 
* while home\_team != away\_team:
* 
* for home\_team == away\_team:
* 
* away\_team = home\_team
* 
* if home\_team != away\_team:
* **Correct**
* You got it! We want to print all possible team pairings but exclude those where a team would play against itself. To do that, we need a conditional that skips the case where both teams are the same.
* The validate\_users function is used by the system to check if a list of users is valid or invalid. A valid user is one that is at least 3 characters long. For example, ['taylor', 'luisa', 'jamaal'] are all valid users. When calling it like in this example, something is not right. Can you figure out what to fix?
* def validate\_users(users):
* for user in users:
* if is\_valid(user):
* print(user + " is valid")
* else:
* print(user + " is invalid")
* validate\_users(["purplecat"])
* RunReset
* Here is your output:
* purplecat is valid
* Nice job! You figured out that you needed to call the
* function with a list instead of just a string!

## Loops Cheat Sheet

## Loops Cheat Sheet

Check out below for a run down of the syntax for while loops and for loops.

### **While Loops**

A while loop executes the body of the loop while the condition remains True.

Syntax:

while condition:

    body

Things to watch out for!

* **Failure to initialize variables.**Make sure all the variables used in the loop’s condition  are initialized before the loop.
* **Unintended infinite loops.**Make sure that the body of the loop modifies the variables used in the condition, so that the loop will eventually end **for all possible values of the variables.**

Typical use:

While loops are mostly used when there’s an unknown number of operations to be performed, and a condition needs to be checked at each iteration.

### **For Loops**

A for loop iterates over a sequence of elements, executing the body of the loop for each element in the sequence.

Syntax:

for variable in sequence

    body

The range() function:

range() generates a sequence of integer numbers. It can take one, two, or three parameters:

* range(n): 0, 1, 2, ... n-1
* range(x,y): x, x+1, x+2, ... y-1
* range(p,q,r): p, p+r, p+2r, p+3r, ... q-1 (if it's a valid increment)

**Common pitfalls:**

* **Forgetting that the upper limit of a range() isn’t included.**
* **Iterating over non-sequences.** Integer numbers aren’t iterable. Strings are iterable letter by letter, but that might not be what you want.

Typical use:

For loops are mostly used when there's a pre-defined sequence or range of numbers to iterate.

### **Break & Continue**

You can interrupt both while and for loops using the break keyword. We normally do this to interrupt a cycle due to a separate condition.

You can use the continue keyword to skip the current iteration and continue with the next one. This is typically used to jump ahead when some of the elements of the sequence aren’t relevant.

If you want to learn more, check out this [wiki page on for loops](https://wiki.python.org/moin/ForLoop).