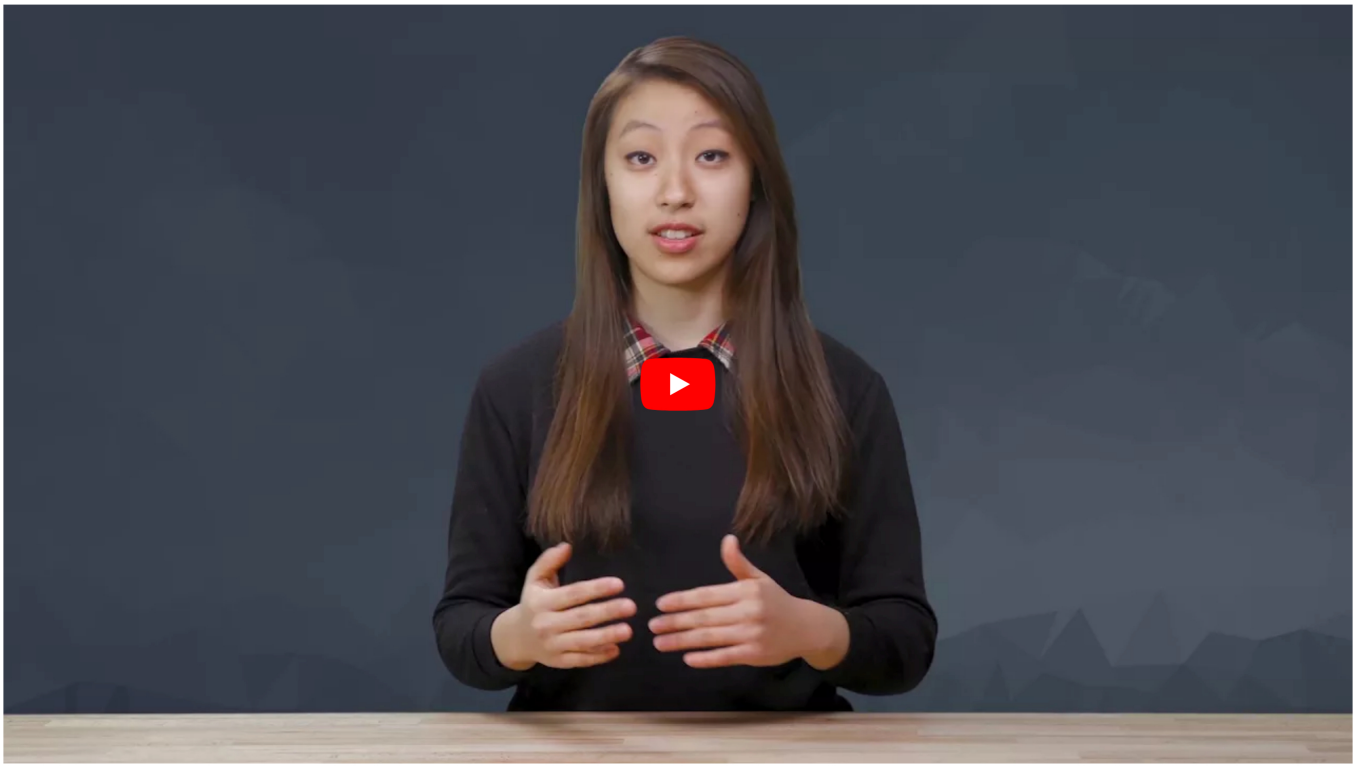
Defining Functions

Before we can start using a function, we first need to define what the function does and what kind of input it may need. The code in the function will only get executed when we **call** or use this function. So functions and function calls provide another way to control the flow of a Python program.

There are two videos on this page, one on defining functions and one on default arguments.



Defining Functions

Example of a function definition:

```
def cylinder_volume(height, radius):
    pi = 3.14159
    return height  *pi*  radius ** 2
```

After defining the cylinder_volume function, we can **call** the function like this.

```
cylinder_volume(10, 3)
```

This is called a **function call** statement.

A function definition includes several important parts.

Function Header

Let's start with the function header, which is the first line of a function definition.

1. The function header always starts with the `def` keyword, which indicates that this is a **function definition**.
2. Then comes the **function name** (here, `cylinder_volume`), which follows the same naming conventions as variables. You can revisit the naming conventions below.
3. Immediately after the name are *parentheses* that may include **arguments** separated by commas (here, `height` and `radius`). Arguments, or **parameters**, are values that are

passed in as **inputs** when the function is called, and are used in the function body. If a function doesn't take arguments, these parentheses are left empty.

4. The header always end with a colon `:`.

Function Body

The rest of the function is contained in the body, which is where the function does its work.

1. The **body** of a function is the code indented after the header line. Here, it's the two lines that define `pi` and `return` the volume.
2. Within this body, we can refer to the **argument variables** and define new variables, which can only be used within these indented lines.
3. The body will often include a `return` statement, which is used to send back an **output value** from the function to the statement that called the function. A `return` statement consists of the `return` keyword followed by an expression that is evaluated to get the output value for the function. If there is no `return` statement, the function simply returns None.

Below, you'll find a code editor where you can experiment with this.

Naming Conventions for Functions

Function names follow the same naming conventions as variables.

1. Only use ordinary letters, numbers and underscores in your function names. They can't have spaces, and need to start with a letter or underscore.
2. **You can't use reserved words or built-in identifiers** that have important purposes in Python, which you'll learn about throughout this course. A list of Python reserved words is described **here**.
3. Try to use descriptive names that can help readers understand what the function does.

Quiz Question

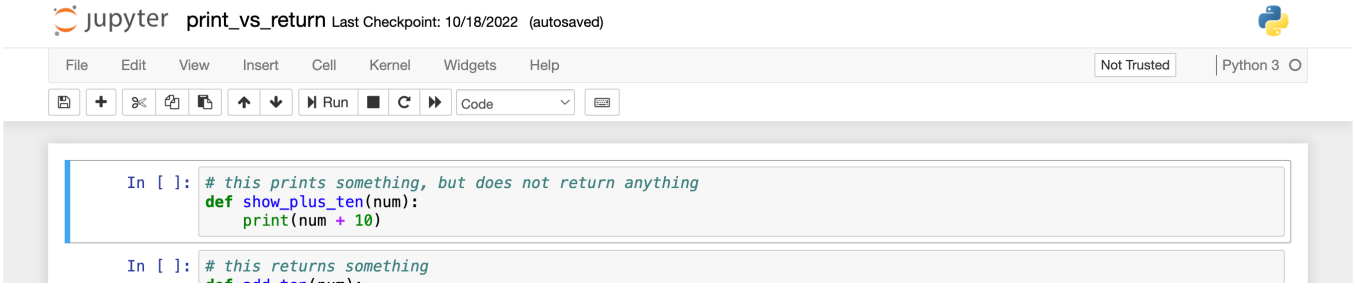
Which of the below are acceptable for a function header in Python? (Select all that apply.)

- [x] def my_function(arg1, arg2):
- [ ] def do_stuff(arg1 arg2):
- [ ] def my function(arg1, arg2)
- [x] def my_function(arg2, arg1, arg4):

Submit

Print vs. Return in Functions

Here are two valid functions. One returns a value and one simply prints a value, without returning anything. Test run this code and experiment to understand the difference.

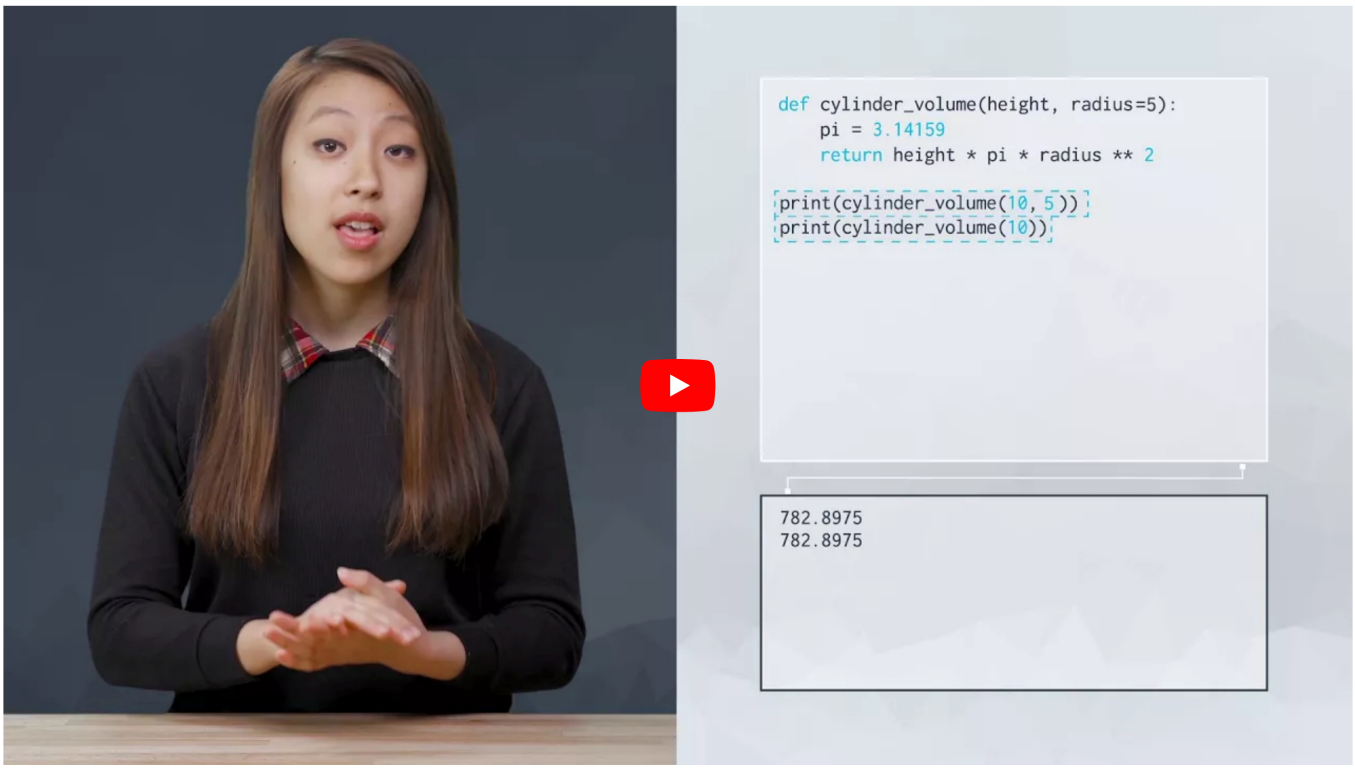


```
def add_ten(num):
    return(num + 10)
```

In [ ]:
```
print('Calling show_plus_ten...')
return_value_1 = show_plus_ten(5)
print('Done calling')
print('This function returned: {}'.format(return_value_1))
```

In [ ]:
```
print('\nCalling add_ten...')
return_value_2 = add_ten(5)
print('Done calling')
print('This function returned: {}'.format(return_value_2))
```

Menu Expand



Default Arguments