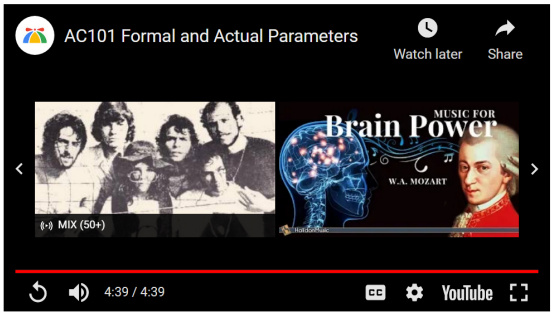
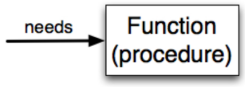
12.4. Function Parameters

Named functions are nice because, once they are defined and we understand what they do, we can refer to them by name and not think too much about what they do. With parameters, functions are even more powerful, because they can do pretty much the same thing on each invocation, but not exactly the same thing. The parameters can cause them to do something a little different.



Activity: 1 -- Video: (goog_function_parms)

The figure below shows this relationship. A function needs certain information to do its work. These values, often called **arguments** or **actual parameters** or **parameter values**, are passed to the function by the user.



This type of diagram is often called a **black-box diagram** because it only states the requirements from the perspective of the user (well, the programmer, but the programmer who uses the function, who may be different than the programmer who created the function). The user must know the name of the function and what arguments need to be passed. The details of how the function works are hidden inside the "black-box".

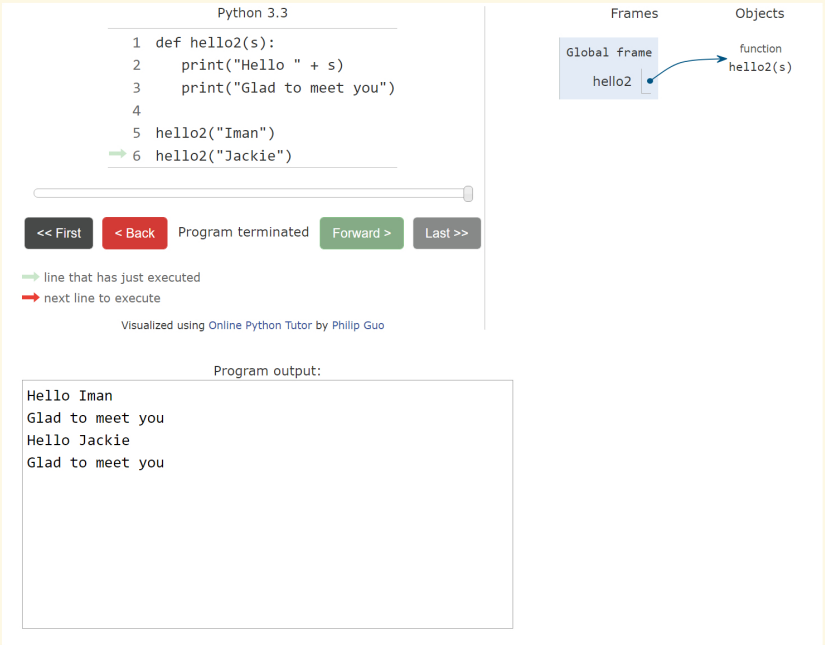
You have already been making function invocations with parameters. For example, when you write `len("abc")` or `len([3, 9, "hello"])`, len is the name of a function, and the value that you put inside the parentheses, the string "abc" or the list [3, 9, "hello"], is a parameter value.

When a function has one or more parameters, the names of the parameters appear in the function definition, and the values to assign to those parameters appear inside the parentheses of the function invocation. Let's look at each of those a little more carefully.

In the definition, the parameter list is sometimes referred to as the **formal parameters** or **parameter names**. These names can be any valid variable name. If there is more than one, they are separated by commas.

In the function invocation, inside the parentheses one value should be provided for each of the parameter names. These values are separated by commas. The values can be specified either directly, or by any python expression including a reference to some other variable name.

That can get kind of confusing, so let's start by looking at a function with just one parameter. The revised hello function personalizes the greeting: the person to greet is specified by the parameter.



Activity: 2 -- CodeLens: (clens11_3_1)

First, notice that hello2 has one formal parameter, s. You can tell that because there is exactly one variable name inside the parentheses on line 1.

Next, notice what happened during Step 2. Control was passed to the function, just like we saw before. But in addition, the variable s was bound to a value, the string "Iman". When it got to Step 7, for the second invocation of the function, s was bound to "Jackie".

Function invocations always work that way. The expression inside the parentheses on the line that invokes the function is evaluated before control is passed to the function. The value is assigned to the corresponding formal parameter. Then, when the code block inside the function is executing, it can refer to that formal parameter and get its value, the value that was 'passed into' the function.

Next Step Reset

```
def hello2(s):
print("Hello " + s)
print("Glad to meet you")

hello2("Nick")
```

```
hello2("Nick")
def hello2(s):
def hello2("Nick"):
print("Hello " + s)
print("Hello " + "Nick") #prints out "hello Nick"
print("Glad to meet you") #prints out "Glad to meet you"
# the function is finished
```

Activity: 3 -- ShowEval (eval11_3_1)

To get a feel for that, let's invoke hello2 using some more complicated expressions. Try some of your own, too.

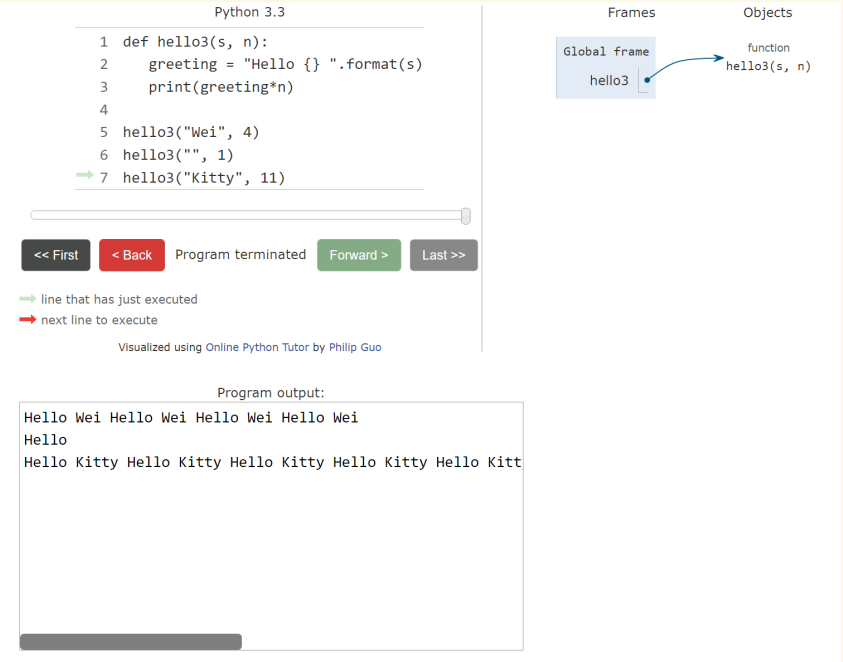
Save & Run Original - 1 of 1 Show in CodeLens

```
1 def hello2(s):
2     print("Hello " + s)
3     print("Glad to meet you")
4
5 hello2("Iman" + " and Jackie")
6 hello2("Class " * 3)
7
```

```
Hello Iman and Jackie
Glad to meet you
Hello Class Class Class
Glad to meet you
```

Activity: 4 -- ActiveCode (ac11_3_1)

Now let's consider a function with two parameters. This version of hello takes a parameter that controls how many times the greeting will be printed.



Activity: 5 -- CodeLens: (clens11_3_2)

At Step 3 of the execution, in the first invocation of hello3, notice that the variable s is bound to the value "Wei" and the variable n is bound to the value 4.

That's how function invocations always work. Each of the expressions, separated by commas, that are inside the parentheses are evaluated to produce values. Then those values are matched up positionally with the formal parameters. The first parameter name is bound to the first value provided. The second

parameter name is bound to the second value provided. And so on.

Check your understanding

func-3-1: Which of the following is a valid function header (first line of a function definition)?

- A. def greet(t):
- B. def greet:
- C. greet(t, n):
- D. def greet(t, n)

Check me | Compare me

✔ A function may take zero or more parameters. In this case it has one.

Activity: 6 -- Multiple Choice (question11_3_1)

func-3-2: What is the name of the following function?

```
def print_many(x, y):
    """Print out string x, y times."""
    for i in range(y):
        print(x)
```

- A. def print_many(x, y):
- B. print_many
- C. print_many(x, y)
- D. Print out string x, y times.

Check me | Compare me

✔ Yes, the name of the function is given after the keyword def and before the list of parameters.

Activity: 7 -- Multiple Choice (question11_3_2)

func-3-3: What are the parameters of the following function?

```
def print_many(x, y):
    """Print out string x, y times."""
    for i in range(y):
        print(x)
```

- A. i
- B. x
- C. x, y
- D. x, y, i

Check me | Compare me

✔ Yes, the function specifies two parameters: x and y.

Activity: 8 -- Multiple Choice (question11_3_3)

func-3-4: Considering the function below, which of the following statements correctly invokes, or calls, this function (i.e., causes it to run)?

```
def print_many(x, y):
   """Print out string x, y times."""
   for i in range(y):
       print(x)

z = 3
```

- A. print_many(x, y)
- B. print_many
- C. print_many("Greetings")
- D. print_many("Greetings", 10):
- E. print_many("Greetings", z)

Check me | Compare me

✔ Since z has the value 3, we have passed in two correct values for this function. "Greetings" will be printed 3 times.

Activity: 9 -- Multiple Choice (question11_3_4)

func-3-5: True or false: A function can be called several times by placing a function call in the body of a for loop.

- A. True
- B. False

Check me | Compare me

✔ Yes, you can call a function multiple times by putting the call in a loop.

Activity: 10 -- Multiple Choice (question11_3_5)

func-3-6: What output will the following code produce?

```
def cyu(s1, s2):
   if len(s1) > len(s2):
      print(s1)
   else:
      print(s2)

cyu("Hello", "Goodbye")
```

- A. Hello
- B. Goodbye
- C. s1
- D. s2

Check me | Compare me

✔ "Goodbye" is longer than "Hello"

Activity: 11 -- Multiple Choice (question11_3_6)

You have attempted 12 of 11 activities on this page

12.3. Function Invocation">
nction Invocation">
12.5. Returning a value from a function">
✔ Completed. Well Don
12.5. Returning a value from a function">Next Section - 12.5. Returning a value from a function

| Back to top