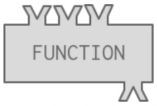
2.4. Function Calls

The Python interpreter can compute new values with function calls. You are familiar with the idea of functions from high school algebra. There you might define a function `f` by specifying how it transforms an input into an output, `f(x) = 3x + 2`. Then, you might write `f(5)` and expect to get the value 17.

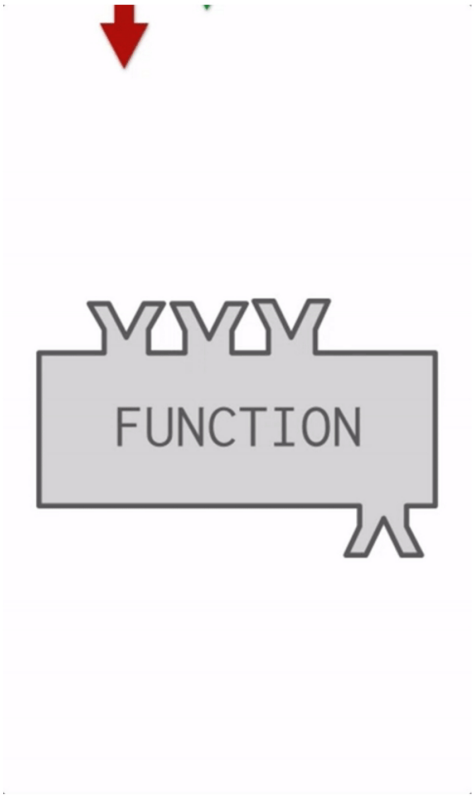
Python adopts a similar syntax for invoking functions. If there is a named function `foo` that takes a single input, we can invoke foo on the value 5 by writing `foo(5)`.

There are many built-in functions available in Python. You'll be seeing some in this chapter and the next couple of chapters.

Functions are like factories that take in some material, do some operation, and then send out the resulting object.



In this case, we refer to the materials as arguments or inputs and the resulting object is referred to as output or return value. This process of taking input, doing something, and then sending back the output is demonstrated in the gif below.



Note

Don't confuse the "output value" of a function with the output window. The output of a function is a Python value and we can never really see the internal representation of a value. But we can draw pictures to help us imagine what values are, or we can print them to see an external representation in the output window.

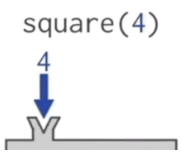
To confuse things even more, `print` is actually a function. All functions produce output values. Only the `print` function causes things to appear in the output window.

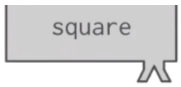
It is also possible for programmers to define new functions in their programs. You will learn how to do that later in the course. For now, you just need to learn how to invoke, or call, a function, and understand that the execution of the function returns a computed value.

Save & Run Show Code Show CodeLens

Activity: 1 -- ActiveCode (ac2_4_1)

We've defined two functions above. The code is hidden so as not to bother you (yet) with how functions are defined. `square` takes a single input parameter, and returns that input multiplied by itself. `sub` takes two input parameters and returns the result of subtracting the second from the first. Obviously, these functions are not particularly useful, since we have the operators `*` and `-` available. But they illustrate how functions work. The visual below illustrates how the `square` function works.





Save & Run 4/17/2021, 10:45:44 AM - 2 of 2 Show in CodeLens

```
print(square(3))
square(5)
print(sub(6, 4))
print(sub(5, 9))

```

```
9
2
-4
```

Activity: 2 -- ActiveCode (ac2_4_2)

Notice that when a function takes more than one input parameter, the inputs are separated by a comma. Also notice that the order of the inputs matters. The value before the comma is treated as the first input, the value after it as the second input.

Again, remember that when Python performs computations, the results are only shown in the output window if there's a print statement that says to do that. In the activecode window above, `square(5)` produces the value 25 but we never get to see that in the output window, because it is not printed.

2.4.1. Function calls as part of complex expressions

Anywhere in an expression that you can write a literal like a number, you can also write a function invocation that produces a number.

For example:

Save & Run 4/17/2021, 10:46:02 AM - 2 of 2 Show in CodeLens

```
print(square(3) + 2)
print(sub(square(3), square(1+1)))

```

```
11
5
```

Activity: 3 -- ActiveCode (ac2_4_3)

Let's take a look at how that last execution unfolds.

Next Step Reset

```
Notice that we always have to resolve the expression inside the innermost
parentheses first, in order to determine what input to provide when calling the
functions.
```

```
print(sub(square(3), square(1+1)))
print(sub(9, square(1+1)))
print(sub(9, square(2)))
print(sub(9, 4))
print(5)
```

Activity: 4 -- ShowEval (se_ac2_4_1a)

2.4.2. Functions are objects; parentheses invoke functions

Remember that earlier we mentioned that some kinds of Python objects don't have a nice printed representation? Functions are themselves just objects. If you tell Python to print the function object, rather than printing the results of invoking the function object, you'll get one of those not-so-nice printed representations.

Just typing the name of the function refers to the function as an object. Typing the name of the function followed by parentheses `()` invokes the function.

Save & Run 4/17/2021, 10:46:35 AM - 2 of 2 Show in CodeLens

```
1 print(square)
2 print(square(3))
3
```

```
<function square>
9
```

Activity: 5 -- ActiveCode (ac2_4_4)

You have attempted 5 of 5 activities on this page

✔ Completed. Well Done!

2.3. Operators and Operands">
erators and Operands">
2.5. Data Types">

| Back to top