KEY_Lesson08_Functions_and_Methods

May 28, 2020

1 Functions and Methods

Functions and methods are things that take an input, do something with that input, and then spit out an output. You've already been using functions and methods!

Examples of functions: - `print`: prints the input out to the screen - `len`: finds the length of the input

Example of a method: - `append`: appends elements to a list

So what's the difference between functions and methods?

- **Functions** can take lots of different object types as input. For example, `print` can take a variety of inputs including a variable, a string, a list, or numbers.
- **Methods** are special cases of functions, so everything that is true for functions is also true for methods (i.e. they can take a variety of input). However, methods *can only be used on one specific type of variable*. For example, append can only be used on lists (and not strings or numbers).

Differences in syntax (how we write it to use it): - The input to **functions** goes *between* parentheses after the name of function - Example: `print(print_input)`

- For **methods**, the variable they operate on comes *before* the name of the method and is followed by a period (`.`). Then, any other input the method might require goes between the parentheses, similar to functions.
- Example: `mylist.append(what_to_append)`

We're going to learn a few more functions and methods here!

Let's start by learning some functions and methods we can use with lists of numbers!

[0]:
```
# make a list of numbers
numbers = [10,2.3,-4,20,14,1,2,0,-3,1,-2,2,2,65.4,3,-23,123,43.1,32,57,32]
```

What function can you use to find the length of this list? Use this function to find the length:

[2]:
```
# get the length of numbers
len(numbers)
```

[2]: 21

If you're not sure how to use a function, you can use the function `help` to get more information about it. Here's an example to find more information about length:

[3]:
```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

It tells you that `len` takes an object and returns the number of items in the object.

To get the object type of `numbers` we can use the function `type`:

[4]:
```
# print the type of numbers
type(numbers)
```

[4]:
```
list
```

Nice! It's a list, just like we expected. What do you think the type is of the first element in `numbers`?

[5]:
```
# print the first element of numbers
print(numbers[0])

# print the type of the first element of numbers
type(numbers[0])
```

```
10
```

[5]:
```
int
```

It's an integer. What about the type of the second element of numbers?

[6]:
```
# print the second element of numbers
print(numbers[1])

# print the type of the second element of numbers
type(numbers[1])
```

```
2.3
```

[6]:
```
float
```

It's a float. This means it has a decimal place (so it's not an integer).

We can also find the absolute value of a number. Print the third element and then print the absolute value of this number using the function `abs`:

[7]:
```
# print the third element of numbers
print(numbers[2])

# print the type of the third element of numbers
print(abs(numbers[2]))
```

```
-4
4
```

Now let's find the minimum of the numbers list using the function `min` and the maximum of the numbers list using the function `max`:

[8]:
```
# print the minimum of the numbers list
print(min(numbers))

# print the maximum of the numbers list
print(max(numbers))
```

```
-23
123
```

We can also find the sum of all of the numbers in `numbers` using the function `sum`:

[9]:
```
# print the sum of numbers
print(sum(numbers))
```

```
377.8
```

What if we want to find the mean of this list? If you remember from math class the formula for calculating a list is as follows:

mean = sum_of_all_values/total_number_of_values

Hint: We can use the functions `sum` and `len` to do this.

[10]:
```
# save the mean of the numbers list to the variable avg
avg = sum(numbers)/len(numbers)

# print avg
print(avg)
```

```
17.99047619047619
```

What if we want to round this number so there aren't so many decimal places? We can use the function `round` to do this:

[11]:
```
# round avg
print(round(avg))
```

```
18
```

What if we want to include 2 decimal places when we round? We can use what's called an *argument* to tell the function that we want to do this. First, let's look at the help description for `round`:

[12]:
```
help(round)
```

```
Help on built-in function round in module builtins:

round(…)
    round(number[, ndigits]) -> number

    Round a number to a given precision in decimal digits (default 0 digits).
    This returns an int when called with one argument, otherwise the
    same type as the number. ndigits may be negative.
```

It says that the default number of digits to round to is zero, but you can also give an argument (i.e. another input to the `round` function) to specify the number of digits. To add an argument, you put a comma after the input and then the argument. Let's try it out!

[13]:
```
# print avg rounded to 2 decimal places
print(round(avg,2))
```

```
17.99
```

Nice job! You just learned about functions in Python! You learned: - What functions and methods do - The difference between functions and methods - How to learn more about a certain function or method (using the `help` function) - New functions: `max`, `min`, `sum`, `abs`, `round` - Functions can take arguments that modify the output