Defining a custom function

INTERMEDIATE PYTHON FOR DEVELOPERS

Jasmin Ludolf
Senior Data Science Content Developer

Calculating the average

```python
# List of preparation times (minutes)
preparation_times = [19.23, 15.67, 48.57, 23.45, 12.06, 34.56, 45.67]

# Calculating average preparation time
average_time = sum(preparation_times) / len(preparation_times)

# Rounding the results
rounded_average_time = round(average_time, 2)
print(average_time)
```

```
28.46
```

When to make a custom function

*Don't Repeat Yourself **(DRY)***

- Considerations for making a custom function:
  - Number of lines
  - Code complexity
  - Frequency of use

Creating a custom function

```python
# Create a custom function to calculate the average value
def
```

Creating a custom function

```
# Create a custom function to calculate the average value
def average
```

Creating a custom function

```
# Create a custom function to calculate the average value
def average(
```

Creating a custom function

```python
# Create a custom function to calculate the average value
def average(values)
```

- `values` (argument) - information the function needs to do its job

Creating a custom function

```python
# Create a custom function to calculate the average value
def average(values):
```

Creating a custom function

```python
# Create a custom function to calculate the average value
def average(values):
    # Calculate the average
    average_value = sum(values) / len(values)
```

Creating a custom function

```
# Create a custom function to calculate the average value
def average(values):
    # Calculate the average
    average_value = sum(values) / len(values)

    # Round the results
    rounded_average = round(average_value, 2)
```

Creating a custom function

```python
# Create a custom function to calculate the average value
def average(values):
    # Calculate the average
    average_value = sum(values) / len(values)

    # Round the results
    rounded_average = round(average_value, 2)

    # Return an output
    return
```

- `average_value` and `rounded_average` are only available within `average()`

Creating a custom function

```
# Create a custom function to calculate the average value
def average(values):
    # Calculate the average
    average_value = sum(values) / len(values)

    # Round the results
    rounded_average = round(average_value, 2)

    # Return rounded_average as an output
    return rounded_average
```

- Documentation is essential

Using a custom function

```python
# List of preparation times (minutes)
preparation_times = [19.23, 15.67, 48.57, 23.45, 12.06, 34.56, 45.67]

# Calculating the average
print(average(preparation_times))
```

```
28.46
```

```python
# List of orders
orders = [12, 8, 10, 9, 15, 21, 16]
print(average(orders))
```

```
12.86
```

Storing a function's output

```
# Calculating the average
print(average(preparation_times))
```

```
28.46
```

```
# Storing average_time
average_time = average(preparation_times)
print(average_time)
```

```
28.46
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Default and keyword arguments

INTERMEDIATE PYTHON FOR DEVELOPERS

Jasmin Ludolf
Senior Data Science Content Developer

Average

```python
# Create a custom function
def average(values):
    # Calculate the average
    average_value = sum(values) / len(values)

    # Round the results
    rounded_average = round(average_value, 2)

    # Return rounded_average as an output
    return rounded_average
```

- `values` = Argument

Arguments

- Values provided to a function or method
    - **Positional**
    - **Keyword**

Positional arguments

- Provide arguments in order, separated by commas

```
# Round pi to 2 digits
print(round(3.1415926535, 2))
```

```
3.14
```

Keyword arguments

- Provide arguments by assigning values to `keywords`
- Useful for interpretation and tracking arguments

```
# Round pi to 2 digits
print(round(number=3.1415926535
```

Keyword arguments

- Provide arguments by assigning values to `keywords`
- Useful for interpretation and tracking arguments

```
# Round pi to 2 digits
print(round(number=3.1415926535, ndigits=2))
```

```
3.14
```

Identifying keyword arguments

```
# Get more information about the help function
print(help(round))
```

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

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise,
    the return value has the same type as the number.  ndigits may be negative.
```

Keyword arguments

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

round(number, ndigits=None)
  Round a number to a given precision in decimal digits.

  The return value is an integer if ndigits is omitted or None. Otherwise
  the return value has the same type as the number. ndigits may be negative.
```

- First argument: `number`
- Second argument: `ndigits`

Default arguments

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

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise,
    the return value has the same type as the number.  ndigits may be negative.
```

- `None` = no value / empty
- Default argument: way of setting a `default` value for an `argument`
- We overwrite `None` to `2`
- Commonly used value - set it using a default argument

Adding an argument

```python
# Create a custom function
def average(values):
    average_value = sum(values) / len(values)
    rounded_average = round(average_value, 2)
    return rounded_average
```

Adding an argument

```python
# Create a custom function
def average(values, rounded=False):
```

Adding an argument

```python
# Create a custom function
def average(values, rounded=False):
    # Round average to two decimal places if rounded is True
    if rounded == True:
        average_value = sum(values) / len(values)
        rounded_average = round(average_value, 2)
        return rounded_average
```

Adding an argument

```
# Create a custom function
def average(values, rounded=False):
    # Round average to two decimal places if rounded is True
    if rounded == True:
        average_value = sum(values) / len(values)
        rounded_average = round(average_value, 2)
        return rounded_average
    # Otherwise, don't round
    else:
        average_value = sum(values) / len(values)
        return average_value
```

Using the modified average() function

```python
# List of preparation times (minutes)
preparation_times = [19.23, 15.67, 48.57, 23.45, 12.06, 34.56, 45.67]
```

Using the modified average() function

```
# Get the average without rounding
print(average(preparation_times, False))
```

```
28.4585714
```

```
# Get the average without rounding
print(average(preparation_times))
```

```
28.4585714
```

Using the modified average() function

```
# Get the rounded average
print(average(values=preparation_times, rounded=True))
```

```
28.46
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Docstrings

INTERMEDIATE PYTHON FOR DEVELOPERS

Jasmin Ludolf
Senior Data Science Content Developer

Docstrings

- String (block of text) describing a function
- Help users understand how to use a function

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

round(number, ndigits=None)
  Round a number to a given precision in decimal digits.

  The return value is an integer if ndigits is omitted or None. Otherwise
  the return value has the same type as the number. ndigits may be negative.
```

Accessing a docstring

```python
# Access information, including the docstring
print(help(round))
```

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

round(number, ndigits=None)
    Round a number to a given precision in decimal digits.

    The return value is an integer if ndigits is omitted or None.  Otherwise
    the return value has the same type as the number.  ndigits may be negative.

None
```

Accessing a docstring

```
# Access only the docstring
print(round
```

Accessing a docstring

```
# Access only the docstring
print(round.
```

Accessing a docstring

```
# Access only the docstring
print(round.__
```

Accessing a docstring

```
# Access only the docstring
print(round.__doc
```

Accessing a docstring

```python
# Access only the docstring
print(round.__doc__)
```

```
Round a number to a given precision in decimal digits.

The return value is an integer if ndigits is omitted or None.  Otherwise
the return value has the same type as the number.  ndigits may be negative.
```

- `.__doc__` : "dunder-doc" attribute

Creating a docstring

```python
def average(values):
    # One-line docstring
    """Find the mean in a sequence of values and round to two decimal places."""
    average_value = sum(values) / len(values)
    rounded_average = round(average_value, 2)
    return rounded_average
```

Accessing the docstring

```python
# Access our docstring
print(average.__doc__)
```

```
Find the mean in a sequence of values and round to two decimal places.
```

Updating a docstring

```python
# Update a function's docstring
average.__doc__ = "Calculate the mean of values in a data structure, rounding the results to 2 digits."

print(help(average))
```

```
Help on function average in module __main__:


average(values)
    Calculate the mean of values in a data structure, rounding the results to 2 digits.
```

Multi-line docstring

```python
def average(values):
    """
    Find the mean in a sequence of values and round to two decimal places.







    """
    average_value = sum(values) / len(values)
    rounded_average = round(average_value, 2)
    return rounded_average
```

Multi-line docstring

```
def average(values):
    """
    Find the mean in a sequence of values and round to two decimal places.

    Args:





    """
    average_value = sum(values) / len(values)
    rounded_average = round(average_value, 2)
    return rounded_average
```

Multi-line docstring

```python
def average(values):
    """
    Find the mean in a sequence of values and round to two decimal places.

    Args:
        values (list): A list of numeric values.




    """
    average_value = sum(values) / len(values)
    rounded_average = round(average_value, 2)
    return rounded_average
```

Multi-line docstring

```python
def average(values):
    """
    Find the mean in a sequence of values and round to two decimal places.

    Args:
        values (list): A list of numeric values.

    Returns:
        rounded_average (float): The mean of values, rounded to two decimal places.
    """
    average_value = sum(values) / len(values)
    rounded_average = round(average_value, 2)
    return rounded_average
```

Accessing the docstring

```python
# Help
print(help(average))
```

```
Help on function average in module __main__:

average(values)
    Find the mean in a sequence of values and round to two decimal places.

        Args:
            values (list): A list of numeric values.

        Returns:
            rounded_average (float): The mean of values, rounded to two decimal places.
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Arbitrary arguments

INTERMEDIATE PYTHON FOR DEVELOPERS

Jasmin Ludolf
Curriculum Manager

Limitations of defined arguments

```python
def average(values):
    """Find the mean in a sequence of values and round to two decimal places."""

    average_value = sum(values) / len(values)
    rounded_average = round(average_value, 2)
    return rounded_average

# Using six arguments
print(average(15, 29, 4, 13, 11, 8))
```

```
TypeError: average() takes 1 positional argument but 6 were given
```

Arbitrary positional arguments

- Docstrings help clarify how to use custom functions
- Arbitrary arguments allow functions to accept **any number** of arguments

```
# Allow any number of positional, non-keyword arguments
def average(*args):
    # Function code remains the same
```

- Conventional naming: `*args`
- Allows a variety of uses while producing expected results!

Using arbitrary positional arguments

```python
# Calling average with six positional arguments
print(average(15, 29, 4, 13, 11, 8))
```

```
13.33
```

Args create a single iterable

- `*` : Convert arguments to a single iterable (tuple)

```
# Calculating across multiple lists
print(average(*[15, 29], *[4, 13], *[11, 8]))
```

```
13.33
```

Arbitrary keyword arguments

```python
# Use arbitrary keyword arguments
def average(**kwargs):
    average_value = sum(kwargs.values()) / len(kwargs.values())
    rounded_average = round(average_value, 2)
    return rounded_average
```

- Arbitrary keyword arguments: `**kwargs`
- `keyword=value`

Using arbitrary keyword arguments

```python
# Calling average with six kwargs
print(average(a=15, b=29, c=4, d=13, e=11, f=8))
```

```
13.33
```

```python
# Calling average with one kwarg
print(average(**{"a":15, "b":29, "c":4, "d":13, "e":11, "f":8}))
```

```
13.33
```

- Each key-value pair in the dictionary is mapped to a keyword argument and value!

Kwargs create a single iterable

```python
# Calling average with three kwargs
print(average(**{"a":15, "b":29}, **{"c":4, "d":13}, **{"e":11, "f":8}))
```

```
13.33
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS