Defining a custom function

INTERMEDIATE PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Calculating the average

```
# Sales variable
sales = [125.97, 84.32, 99.78, 154.21, 78.50, 83.67, 111.13]

# Calculating average sales
average_sales = sum(sales) / len(sales)

# Rounding the results
rounded_average_sales = round(average_sales, 2)
print(rounded_average_sales)
```

```
105.37
```

Create a custom function

- Considerations for making a custom function:
  - Number of lines
  - Code complexity
  - Frequency of use
  - *Don't Repeat Yourself* ***(DRY)***

Creating a custom function

```
# 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

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

Creating a custom function

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

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

```
# 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
```

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
```

Returning a calculation

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

    # Return the rounded results
    return round(average_value, 2)
```

Using a custom function

```
sales = [125.97, 84.32, 99.78, 154.21, 78.50, 83.67, 111.13]


# Calculating the average
average(sales)
```

```
105.37
```

Storing a function's output

```
# Calculating the average
average(sales)
```

```
105.37
```

```
# Storing average_sales
average_sales = average(sales)
```

```
print(average_sales)
```

```
105.37
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Default and keyword arguments

INTERMEDIATE PYTHON FOR DEVELOPERS

George Boorman
Curriculum Manager, DataCamp

datacamp

Average

```
# 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

- Arguments are values provided to a function or method
- Functions and methods have two types of arguments:
  - **Positional**
  - **Keyword**

Positional arguments

- Provide arguments in order, separated by commas

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

```
3.14
```

Keyword arguments

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

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

Keyword arguments

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

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

```
3.14
```

Identifying keyword arguments

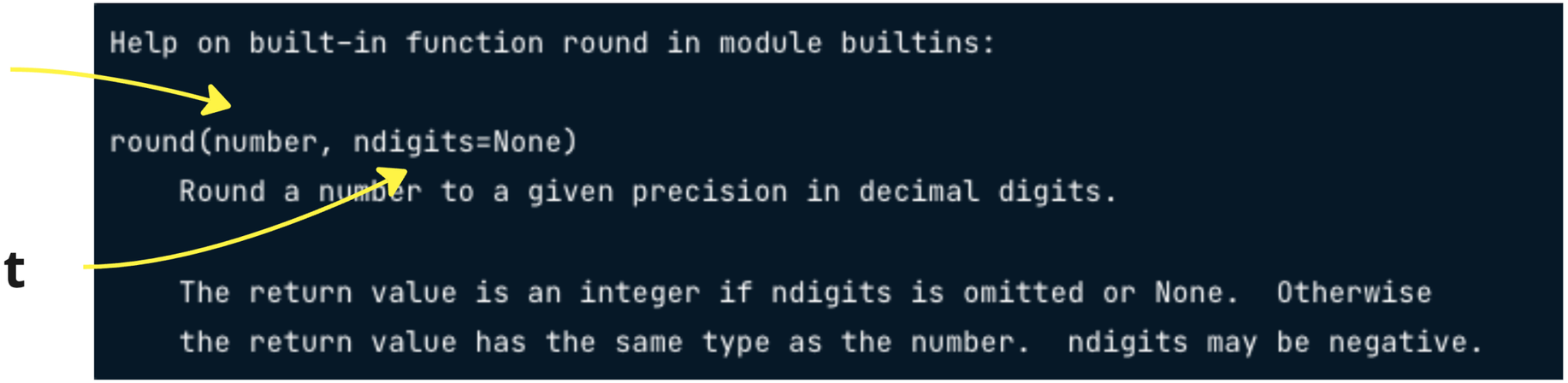
```
# Get more information about the help function
help(round)
```

```
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

First argument

Second argument



```
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.
```

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`
  - Otherwise, the result is an `int`

Why have default arguments?

- Helps us think about likely uses for our function
  - Commonly used value - set it using a default argument

- Potentially reduces code for users (if they stick with default values)

- Maintains flexibility

Adding an argument

```
# 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

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

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
```

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

```
sales = [125.97, 84.32, 99.78, 154.21, 78.50, 83.67, 111.13]
```

Using the modified average() function

```
# Get the average without rounding
average(sales, False)
```

```
105.36857142857141
```

```
# Get the average without rounding
average(sales)
```

```
105.36857142857141
```

Using the modified average() function

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

```
105.37
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Docstrings

INTERMEDIATE PYTHON FOR DEVELOPERS

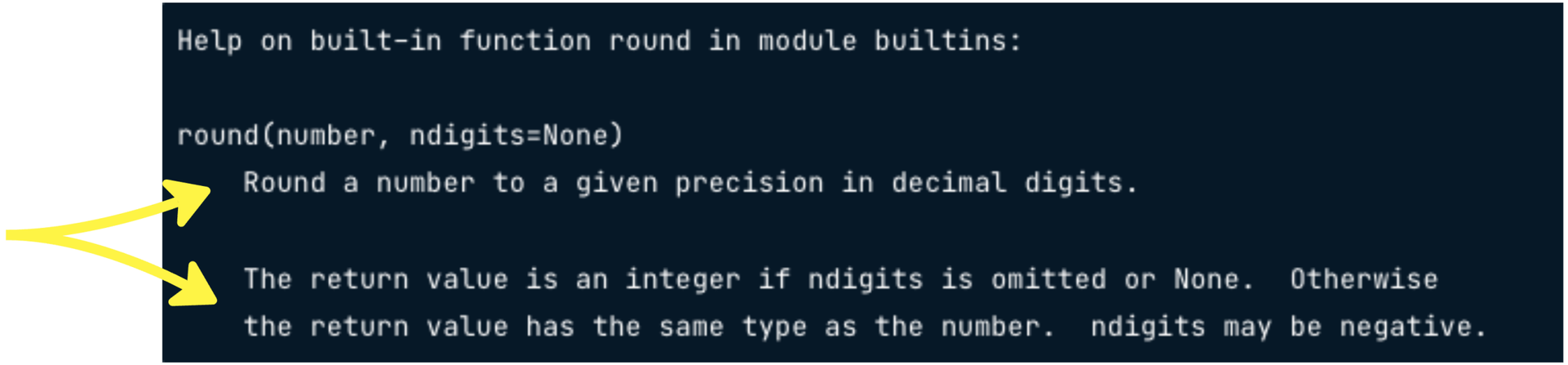
George Boorman
Curriculum Manager, DataCamp

datacamp

Docstrings

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

Docstring



```
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

```
# Access information including the docstring
help(round)
```

Accessing a docstring

```
# Access only the docstring
round
```

Accessing a docstring

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

Accessing a docstring

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

Accessing a docstring

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

Accessing a docstring

```
# Access only the docstring
round.__doc__
```

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

Accessing a docstring

```
# Access only the docstring
round.__doc__
```

```
'Round a number to a given precision in decimal digits.\n\nThe return value is an
integer if ndigits is omitted or None.  Otherwise\nthe return value has the same
type as the number.  ndigits may be negative.'
```

```
help(round)
```

```
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.
```

Creating a docstring

```
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

```
# Access our docstring
average.__doc__
```

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

Updating a docstring

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

Multi-line docstring

```
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

```
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

```
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

```
# Help
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

George Boorman
Curriculum Manager

datacamp

Limitations of defined arguments

```
def average(values):
    # Calculate the average
    average_value = sum(values) / len(values)

    # Return the rounded results
    return round(average_value, 2)

# Using six arguments
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

```
# Calling average with six positional arguments
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
average(*[15, 29], *[4, 13], *[11, 8])
```

```
13.33
```

Arbitrary keyword arguments

```
# 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

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

```
13.33
```

```
# Calling average with one kwarg
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

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

```
13.33
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS