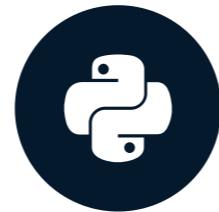


Lambda functions

INTERMEDIATE PYTHON FOR DEVELOPERS



Jasmin Ludolf

Senior Data Science Content Developer

Simple functions

```
def average(values):  
    average_value = sum(values) / len(values)  
    return average_value
```

Lambda functions

- `Lambda` keyword
 - Represents an *anonymous function*

Lambda

Lambda functions

- `Lambda` keyword
 - Represents an *anonymous function*

Lambda arguments

Lambda functions

- `Lambda` keyword
 - Represents an *anonymous function*

Lambda arguments:

Lambda functions

- `lambda` keyword
 - Represents an *anonymous function*

Lambda arguments: expression

- Convention is to use `x` for a single argument
- The `expression` is the equivalent of the function body
- No `return` statement is required
- Can be stored as a variable

Creating a lambda function

```
# Lambda average function  
print(lambda x: sum(x) / len(x))
```

```
<function <lambda> at 0x7f11ab813d80>
```

```
# Custom average function  
def average(x):  
    return sum(x) / len(x)  
  
print(average)
```

```
<function average at 0x7f11ab813ec0>
```

Using lambda functions

```
# Get the average  
(lambda x: sum(x) / len(x))
```

Using lambda functions

```
# Get the average  
(lambda x: sum(x) / len(x))([3, 6, 9])
```

```
# Print the average  
print((lambda x: sum(x) / len(x))([3, 6, 9]))
```

6.0

Storing and calling a lambda function

```
# Store Lambda function as a variable  
average = Lambda x: sum(x) / len(x)
```

```
# Call the average function  
print(average([3, 6, 9]))
```

6.0

Multiple parameters

```
# Lambda function with two arguments  
power = lambda x, y: x**y
```

```
# Raise 2 to the power of 3  
print(power(2, 3))
```

8

Lambda functions with iterables

- `map()` applies a function to **all** elements in an iterable

```
names = ["john", "sally", "leah"]
# Apply a lambda function inside map()
capitalize = map(lambda x: x.capitalize(), names)
print(capitalize)
```

```
<map object at 0x7fb200529c10>
```

```
# Convert to a list
print(list(capitalize))
```

```
['John', 'Sally', 'Leah']
```

Custom vs. lambda functions

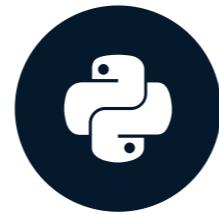
Scenario	Function Type
Complex task	Custom
Same task several times	Custom
Performed once	Lambda
Simple task	Lambda

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Introduction to errors

INTERMEDIATE PYTHON FOR DEVELOPERS

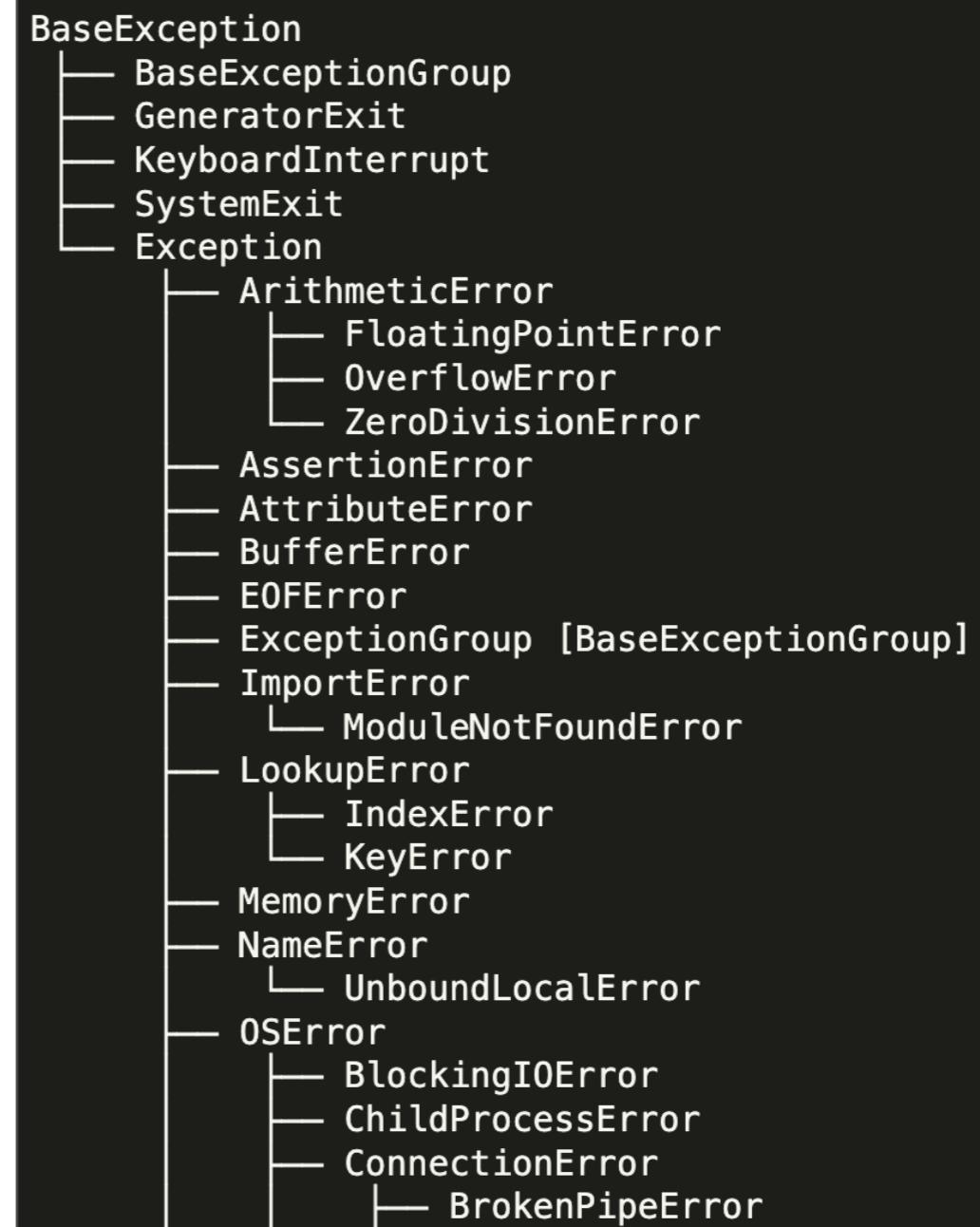


Jasmin Ludolf

Senior Data Science Content Developer

What is an error?

- Code that violates a rule
- Error = Exception
- Cause our program to terminate!



¹ <https://docs.python.org/3/library/exceptions.html#exception-hierarchy>

TypeError

- Incompatible data type

```
"Hello" + 5
```

```
-----  
TypeError  
Cell In[1], line 1  
----> 1 "Hello" + 5  
  
TypeError: can only concatenate str (not "int") to str
```

ValueError

```
float("Hello")
```

```
-----  
ValueError  
Cell In[2], line 1  
----> 1 float("Hello")  
  
ValueError: could not convert string to float: 'Hello'
```

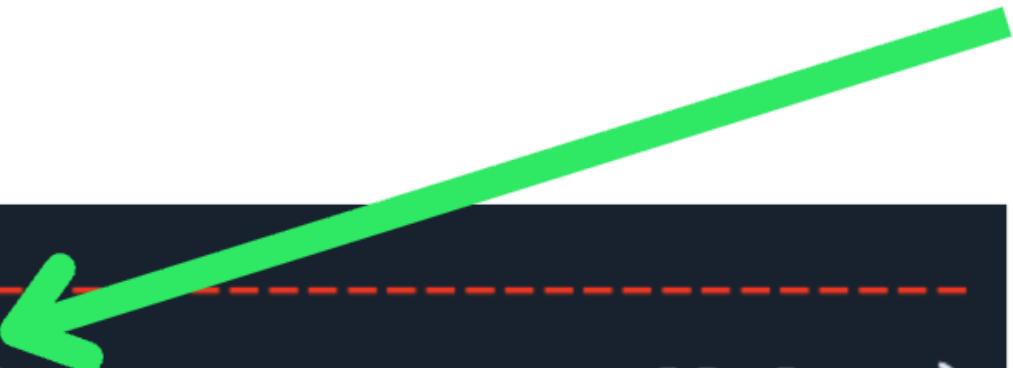
```
float("2")
```

```
2.0
```

- The value is not within an acceptable range

Tracebacks

```
-----  
ValueError  
Cell In[2], line 1  
----> 1 float("Hello")  
  
ValueError: could not convert string to float: 'Hello'
```



Tracebacks

```
-----  
ValueError  
Cell In[2], line 1  
----> 1 float("Hello")  
  
ValueError: could not convert string to float: 'Hello'
```



Tracebacks

```
-----  
ValueError  
Cell In[2], line 1  
----> 1 float("Hello")  
  
ValueError: could not convert string to float: 'Hello'  
-----  
Traceback (most recent call last)
```



Code in packages

- Packages contain other people's code, e.g., custom functions
- Known as source code
- `pip install <package>` downloads source code to our local environment
- The pandas' `pd.read_csv()` function executes the code written for that custom function behind the scenes

Tracebacks from packages

```
# Import pandas package
import pandas as pd

# Create pandas DataFrame
products = pd.DataFrame({"ID": "ABC1",
                          "price": 29.99})

# Try to access the non-existent "tag" column
products["tag"]
```

Tracebacks from packages

```
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3803, in Index.get_loc(self, key, method, tolerance)
    3802     try:
-> 3803         return self._engine.get_loc(casted_key)
    3804     except KeyError as err:
        ...
File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:138, in pandas._libs.index.IndexEngine.get_loc()
File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:165, in pandas._libs.index.IndexEngine.get_loc()
File pandas/_libs/hashtable_class_helper.pxi:5745, in pandas._libs.hashtable.PyObjectHashTable.get_item()
File pandas/_libs/hashtable_class_helper.pxi:5753, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'tag'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[5], line 9
    5 products = pd.DataFrame({ "ID": ["ABC1", "ABC2", "ABC3"],
    6                      "price": [29.99, 39.95, 51.25] })
    8 # Try to access the non-existent "tag" column
-> 9 products["tag"]

File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3804, in DataFrame.__getitem__(self, key)
    3802     if self.columns.nlevels > 1:
    3803         return self._getitem_multilevel(key)
-> 3804     indexer = self.columns.get_loc(key)
    3805     if is_integer(indexer):
    3806         indexer = [indexer]

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3805, in Index.get_loc(self, key, method, tolerance)
    3803     return self._engine.get_loc(casted_key)
    3804     except KeyError as err:
-> 3805         raise KeyError(key) from err
    3806     except TypeError:
    3807         # If we have a listlike key, _check_indexing_error will raise
    3808         # InvalidIndexError. Otherwise we fall through and re-raise
    3809         # the TypeError.
    3810         self._check_indexing_error(key)

KeyError: 'tag'
```

Tracebacks from packages

```
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3803, in Index.get_loc(self, key, method, tolerance)
3802     try:
-> 3803         return self._engine.get_loc(casted_key)
3804     except KeyError as err:
3805         if_tolerance_re.match(str(err)):
3806             return self._engine.get_indexer([key], tolerance=tolerance)
3807         else:
3808             raise
3809
3810     if method == 'pad':
3811         return self._engine.get_indexer([key], method='pad')
3812     else:
3813         return self._engine.get_indexer([key], method=method)
3814
3815     raise KeyError(f"Key {key} not found in {self.__class__.__name__}.")
3816
3817 pandas/_libs/index.pyx:138: in pandas._libs.index.IndexEngine.get_loc()
3818
3819 pandas/_libs/index.pyx:165: in pandas._libs.index.IndexEngine.get_loc()
3820
3821 pandas/_libs/hashtable_class_helper.pxi:5745, in pandas._libs.hashtable.PyObjectHashTable.get_item()
3822
3823 pandas/_libs/hashtable_class_helper.pxi:5753, in pandas._libs.hashtable.PyObjectHashTable.get_item()
3824
3825 KeyError: 'tag'
```

The above exception was the direct cause of the following exception:

Tracebacks from packages

```
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3803, in Index.get_loc(self, key, method, tolerance)
3802     try:
-> 3803         return self._engine.get_loc(casted_key)
3804     except KeyError as err:
3805         if_tolerance_re.match(str(err)):
3806             return self._engine.get_loc(casted_key)
3807         else:
3808             raise
3809
3810     if method == 'pad':
3811         return self._get_loc_with_pad(key)
3812
3813     if tolerance is not None:
3814         if tolerance < 0:
3815             raise ValueError('tolerance must be non-negative')
3816
3817         if tolerance > 1:
3818             tolerance = 1
3819
3820         if tolerance > 0.5:
3821             tolerance = 0.5
3822
3823         if tolerance < 0.5:
3824             tolerance = 0.5
3825
3826         if tolerance < 0.5:
3827             tolerance = 0.5
3828
3829         if tolerance < 0.5:
3830             tolerance = 0.5
3831
3832         if tolerance < 0.5:
3833             tolerance = 0.5
3834
3835         if tolerance < 0.5:
3836             tolerance = 0.5
3837
3838         if tolerance < 0.5:
3839             tolerance = 0.5
3840
3841         if tolerance < 0.5:
3842             tolerance = 0.5
3843
3844         if tolerance < 0.5:
3845             tolerance = 0.5
3846
3847         if tolerance < 0.5:
3848             tolerance = 0.5
3849
3850         if tolerance < 0.5:
3851             tolerance = 0.5
3852
3853         if tolerance < 0.5:
3854             tolerance = 0.5
3855
3856         if tolerance < 0.5:
3857             tolerance = 0.5
3858
3859         if tolerance < 0.5:
3860             tolerance = 0.5
3861
3862         if tolerance < 0.5:
3863             tolerance = 0.5
3864
3865         if tolerance < 0.5:
3866             tolerance = 0.5
3867
3868         if tolerance < 0.5:
3869             tolerance = 0.5
3870
3871         if tolerance < 0.5:
3872             tolerance = 0.5
3873
3874         if tolerance < 0.5:
3875             tolerance = 0.5
3876
3877         if tolerance < 0.5:
3878             tolerance = 0.5
3879
3880         if tolerance < 0.5:
3881             tolerance = 0.5
3882
3883         if tolerance < 0.5:
3884             tolerance = 0.5
3885
3886         if tolerance < 0.5:
3887             tolerance = 0.5
3888
3889         if tolerance < 0.5:
3890             tolerance = 0.5
3891
3892         if tolerance < 0.5:
3893             tolerance = 0.5
3894
3895         if tolerance < 0.5:
3896             tolerance = 0.5
3897
3898         if tolerance < 0.5:
3899             tolerance = 0.5
3900
3901         if tolerance < 0.5:
3902             tolerance = 0.5
3903
3904         if tolerance < 0.5:
3905             tolerance = 0.5
3906
3907         if tolerance < 0.5:
3908             tolerance = 0.5
3909
3910         if tolerance < 0.5:
3911             tolerance = 0.5
3912
3913         if tolerance < 0.5:
3914             tolerance = 0.5
3915
3916         if tolerance < 0.5:
3917             tolerance = 0.5
3918
3919         if tolerance < 0.5:
3920             tolerance = 0.5
3921
3922         if tolerance < 0.5:
3923             tolerance = 0.5
3924
3925         if tolerance < 0.5:
3926             tolerance = 0.5
3927
3928         if tolerance < 0.5:
3929             tolerance = 0.5
3930
3931         if tolerance < 0.5:
3932             tolerance = 0.5
3933
3934         if tolerance < 0.5:
3935             tolerance = 0.5
3936
3937         if tolerance < 0.5:
3938             tolerance = 0.5
3939
3940         if tolerance < 0.5:
3941             tolerance = 0.5
3942
3943         if tolerance < 0.5:
3944             tolerance = 0.5
3945
3946         if tolerance < 0.5:
3947             tolerance = 0.5
3948
3949         if tolerance < 0.5:
3950             tolerance = 0.5
3951
3952         if tolerance < 0.5:
3953             tolerance = 0.5
3954
3955         if tolerance < 0.5:
3956             tolerance = 0.5
3957
3958         if tolerance < 0.5:
3959             tolerance = 0.5
3960
3961         if tolerance < 0.5:
3962             tolerance = 0.5
3963
3964         if tolerance < 0.5:
3965             tolerance = 0.5
3966
3967         if tolerance < 0.5:
3968             tolerance = 0.5
3969
3970         if tolerance < 0.5:
3971             tolerance = 0.5
3972
3973         if tolerance < 0.5:
3974             tolerance = 0.5
3975
3976         if tolerance < 0.5:
3977             tolerance = 0.5
3978
3979         if tolerance < 0.5:
3980             tolerance = 0.5
3981
3982         if tolerance < 0.5:
3983             tolerance = 0.5
3984
3985         if tolerance < 0.5:
3986             tolerance = 0.5
3987
3988         if tolerance < 0.5:
3989             tolerance = 0.5
3990
3991         if tolerance < 0.5:
3992             tolerance = 0.5
3993
3994         if tolerance < 0.5:
3995             tolerance = 0.5
3996
3997         if tolerance < 0.5:
3998             tolerance = 0.5
3999
3999: KeyError: 'tag'
```

The above exception was the direct cause of the following exception:

Tracebacks from packages

The above exception was the direct cause of the following exception:

Tracebacks from packages

```
KeyError                                  Traceback (most recent call last)
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3803, in Index.get_loc(self, key, method, tolerance)
3802     try:
-> 3803         return self._engine.get_loc(casted_key)
3804     except KeyError as err:
3805
File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:138, in pandas._libs.index.IndexEngine.get_loc()
3806
File /usr/local/lib/python3.8/dist-packages/pandas/_libs/index.pyx:165, in pandas._libs.index.IndexEngine.get_loc()
3807
File pandas/_libs/hashtable_class_helper.pxi:5745, in pandas._libs.hashtable.PyObjectHashTable.get_item()
3808
File pandas/_libs/hashtable_class_helper.pxi:5753, in pandas._libs.hashtable.PyObjectHashTable.get_item()
3809
KeyError: 'tag'
The above exception was the direct cause of the following exception:
```

Tracebacks from packages

```
KeyError                                  Traceback (most recent call last)
Cell In[5], line 9
  5 products = pd.DataFrame({"ID": ["ABC1", "ABC2", "ABC3"],
  6                 "price": [29.99, 39.95, 51.25]})
  8 # Try to access the non-existent "tag" column
----> 9 products["tag"]
```

Tracebacks from packages

```
File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3804 in DataFrame.__getitem__(self, key)
 3802 if self.columns.nlevels > 1:
 3803     return self._getitem_multilevel(key)
-> 3804 indexer = self.columns.get_loc(key)
 3805 if is_integer(indexer):
 3806     indexer = [indexer]

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3805, in Index.get_loc(self, key, method, tolerance)
 3803     return self._engine.get_loc(casted_key)
 3804 except KeyError as err:
-> 3805     raise KeyError(key) from err
 3806 except TypeError:
 3807     # If we have a listlike key, _check_indexing_error will raise
 3808     # InvalidIndexError. Otherwise we fall through and re-raise
 3809     # the TypeError.
 3810     self._check_indexing_error(key)

KeyError: 'tag'
```

Tracebacks from packages

```
File /usr/local/lib/python3.8/dist-packages/pandas/core/frame.py:3804, in DataFrame.__getitem__(self, key)
3802 if self.columns.nlevels > 1:
3803     return self._getitem_multilevel(key)
-> 3804 indexer = self.columns.get_loc(key)
3805 if is_integer(indexer):
3806     indexer = [indexer]

File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3805, in Index.get_loc(self, key, method, tolerance)
3803     return self._engine.get_loc(casted_key)
3804 except KeyError as err:
-> 3805     raise KeyError(key) from err
3806 except TypeError:
3807     # If we have a listlike key, _check_indexing_error will raise
3808     # InvalidIndexError. Otherwise we fall through and re-raise
3809     # the TypeError.
3810     self._check_indexing_error(key)

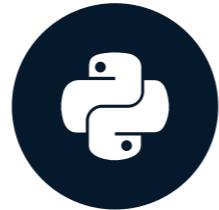
KeyError: 'tag'
```

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Error handling

INTERMEDIATE PYTHON FOR DEVELOPERS



Jasmin Ludolf

Senior Data Science Content Developer

Pandas traceback

```
File /usr/local/lib/python3.8/dist-packages/pandas/core/indexes/base.py:3805, in Index.get_loc(self, key, method, tolerance)
3803     return self._engine.get_loc(casted_key)
3804 except KeyError as err:
-> 3805     raise KeyError(key) from err
3806 except TypeError:
3807     # If we have a listlike key, _check_indexing_error will raise
3808     # InvalidIndexError. Otherwise we fall through and re-raise
3809     # the TypeError.
3810     self._check_indexing_error(key)

KeyError: 'tag'
```

- `except`, `raise`
- Try to anticipate how errors might occur

Design-thinking

- How might people use our custom function?
- Test different approaches
- Find what errors might occur



Error handling in custom functions

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

Where might they go wrong?

- `average()` expects a list or set
- Provide more than one argument □
- Use the wrong data type □

Where might they go wrong?

```
sales_dict = {"cust_id": ["JL93", "MT12", "IY64"],  
              "order_value": [43.21, 68.70, 82.19]}  
  
print(average(sales_dict))
```

```
-----  
TypeError                                                 Traceback (most recent call last)  
Cell In[5], line 4  
      1 sales_dict = {"cust_id": ["JL93", "MT12", "IY64"],  
      2                 "order_value": [43.21, 68.70, 82.19]}  
----> 4 average(sales_dict)  
  
Cell In[4], line 3, in average(values)  
      1 def average(values):  
      2     # Calculate the average  
----> 3     average_value = sum(values) / len(values)  
      4     return average_value  
  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Previous error-handling

- Control flow `if`, `elif`, `else`
- Docstrings

try-except

```
def average(values):
    try:
        # Code that might cause an error
        average_value = sum(values) / len(values)
        return average_value
    except:
        # Code to run if an error occurs
        print("average() accepts a list or set. Please provide a correct data type.")
```

raise

```
def average(values):
    # Check data type
    if type(values) in (list, set):
        # Run if appropriate data type was used
        average_value = sum(values) / len(values)
    return average_value
```

raise

```
def average(values):
    # Check data type
    if type(values) in (list, set):
        # Run if appropriate data type was used
        average_value = sum(values) / len(values)
        return average_value
    else:
        # Run if an Exception occurs
        raise
```

raise TypeError

```
def average(values):
    # Check data type
    if type(values) in (list, set):
        # Run if appropriate data type was used
        average_value = sum(values) / len(values)
        return average_value
    else:
        # Run if an Exception occurs
        raise TypeError("average() accepts a list or set, please provide a correct data type.")
```

raise TypeError output

```
print(average(sales_dict))
```

```
-----
TypeError                                         Traceback (most recent call last)
Cell In[19], line 11
    7     else:
    8         # Run if an Exception occurs
    9         raise TypeError("average() accepts a list or set, please provide a correct data type.")
--> 11 average(sales_dict)

Cell In[19], line 9, in average(values)
    6     return average_value
    7 else:
    8     # Run if an Exception occurs
--> 9     raise TypeError("average() accepts a list or set, please provide a correct data type.")

TypeError: average() accepts a list or set, please provide a correct data type.
```

try-except vs. raise

try - except

- Avoid errors being produced
- Still execute subsequent code

raise

- Will produce an error
- Avoid executing subsequent code

Let's practice!

INTERMEDIATE PYTHON FOR DEVELOPERS

Congratulations

INTERMEDIATE PYTHON FOR DEVELOPERS



Jasmin Ludolf

Senior Data Science Content Developer

Chapter 1 recap

- Built-in functions:
 - `print()` , `help()` , `type()`
 - `max()` , `min()` , `sum()`
 - `len()` , `round()` , `sorted()`
- Modules:
 - `string` , `os`
- Packages:
 - `pandas`

Chapter 2 recap

```
# 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
```

Chapter 2 recap

```
# 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
```

Chapter 2 recap

```
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
```

Chapter 2 recap

```
# Use arbitrary positional arguments
def average(*args):
    average_value = sum(args) / len(args)
    rounded_average = round(average_value, 2)
    return rounded_average

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

Chapter 3 recap

- Lambda arguments: expression

```
names = ["john", "sally", "leah"]

# Apply a lambda function inside map()
capitalize = map(lambda x: x.capitalize(), names)

# Convert to a list
print(list(capitalize))
```

```
['John', 'Sally', 'Leah']
```

Chapter 3 recap

```
-----  
ValueError
```

```
Cell In[2], line 1
```

```
----> 1 float("Hello")
```

```
Traceback (most recent call last)
```

```
ValueError: could not convert string to float: 'Hello'
```

- try
- except
- raise

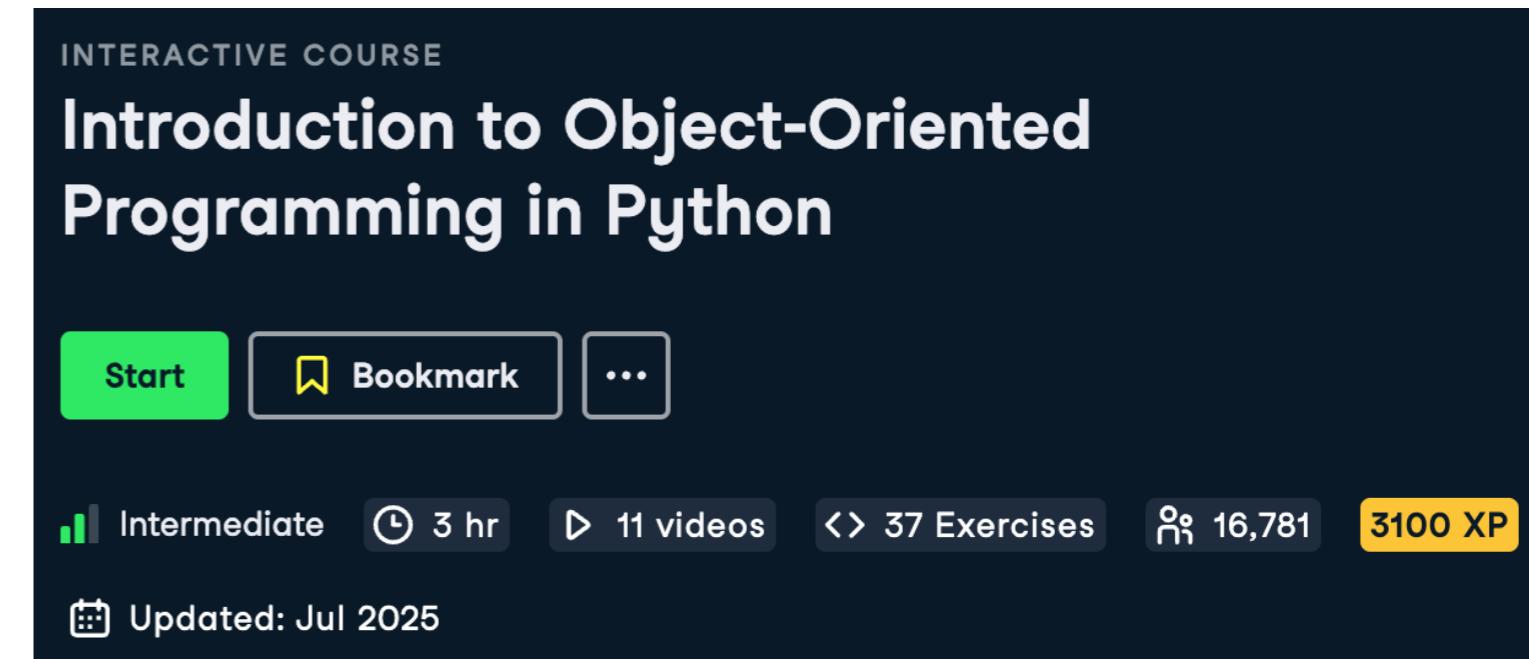
Next steps

- More built-in functions:

- `zip()`
- `input()`
- `reduce()`
- `filter()`

- More packages and modules:

- `time`
- `venv`
- `requests`
- `fastapi`



- Crucial for building software at scale

Congratulations!

INTERMEDIATE PYTHON FOR DEVELOPERS