Programming in Python

MMM001 - Data Engineering

https://github.com/chbrandt/MMM001

Carlos H Brandt

c.brandt@jacobs-university.de

Table of Contents

- Exceptions
  - try/except
  - raise
- Data I/O
  - stdin/stdout/stderr
  - Text files read/write
- Laboratory

Exceptions and Errors

- *Exceptions* are errors detected during software execution: Whenever Python cannot handle an operation, an *error* or *exception is thrown*.
- If exceptions are not handled (properly) the running software *crashes* (or become inconsistent).
- Python provides a mechanism to handle exceptions:

```
(...)
try:
    # do something risky
except:
    # handle eventual error
(...)
```

Exceptions and Errors

- Examples

```
>>> 1/0

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

ZeroDivisionError: division by zero
```

```
>>> bla

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

NameError: name 'bla' is not defined
```

- The error message is indicated in the last line of the *Traceback stack*.
- The stack traceback indicates the origin, and route, of the error.

Built-in Exceptions

- Python provides a set of built-in exceptions
  - https://docs.python.org/3/library/exceptions.html#bltin-exceptions
- Developers can also define their own exceptions
- Example of exceptions:

- IndexError
  - A sequence subscript is out of range
- KeyError
  - A mapping (dictionary) key is not found
- TypeError
  - An operation or function is applied to an object of inappropriate type
- UnboundLocalError
  - No value has been bound to a variable
- FileNotFoundError
  - A file or directory doesn’t exist
- ValueError
  - An argument that has the right type but an inappropriate value

Exceptions and Errors

- `try` statements have one or more `except` blocks, handling different errors

```
(...)
try:
    # do something risky
except <Error>:
    # handle Error
except <AnotherError>:
    # handle AnotherError
except:
    # handle any error
(...)
```

Exceptions and Errors

- `try` statements have one or more `except` blocks, handling different errors
- Possibly a `else` clause

```
(...)
try:
    # do something risky
except:
    # handle error
else:
    # continue doing "something"
(...)
```

Exceptions and Errors

- `try` statements have one or more `except` blocks, handling different errors
- Possibly a `finally` clause

```
(...)
try:
    # do something risky
except:
    # handle error
finally:
    # do something after all
(...)
```

Exceptions and Errors

```
(...)
try:
    # do something risky
except:
    # handle error
else:
    # continue doing "something"
finally:
    # do something after all
(...)
```

Raise exceptions

- Exceptions can be raised with the *raise* statement

```
def foo():
    raise NotImplementedError
```

```
def foo(a):
    if a < 0:
        raise ValueError("Expected a positive argument")
```

Data I/O

- Input/Output of data into a Python application is done (fundamentally) through the user input from the command-line, environment variables, and files
- Input (interactively) from the command-line is handled by the `input` function
- Output to the terminal/prompt is handled by the `print` function

```
>>> var = input('Say something: ')
>>> print(var)
```

Data I/O

- Input/Output of data into a Python application is done (fundamentally) through the user input from the command-line, environment variables, and files
- Environment variables can be read through module `os`: `os.environ`

```
>>> import os
>>> print(os.environ)
```

Data I/O

- Input/Output of data into a Python application is done (fundamentally) through the user input from the command-line, environment variables, and files
- Files are read/write with the open function
  - Once open, files need to be closed after use!

```
>>> f = open('file.txt', 'w')
>>> print('Heyho', file=f)
>>> f.close()
```

Data I/O

- Input/Output of data into a Python application is done (fundamentally) through the user input from the command-line, environment variables, and files
- Files are read/write with the `open` function
  - Once open, files need to be closed after use!

```
>>> with open('file2.txt', 'w') as f:
        print('Yep', file=f)
>>>
```

- `with` is called a *context manager*

Data I/O

- A File object provides methods for reading its lines and "walking" through it
- Suppose there is a file name "myfile.txt". This file can be read line-by-line with its readlines method in a loop:

```
>>> with open('myfile.txt', 'r') as f:
        for line in f.readlines():
            print(line)
>>>
```

References

- Erros and Exceptions: https://docs.python.org/3/tutorial/errors.html
- Built-in: https://docs.python.org/3/library/exceptions.html#bltin-exceptions
- 'with': https://docs.python.org/3/reference/compound_stmts.html#with
- Input/Output: https://docs.python.org/3/tutorial/inputoutput.html
-

Python modules format

Let's apply a few standards to our Python source code -- *i.e.*, module/scripts.

- Each exercise has its own module/script.
- Every module should start with a big """docstring""" explaining the purpose of that file: what is the content, or solution being implemented there.
    - In this docstring, a line starting with '**@author:**' is used for your name
- Each function or class or method should provide a """docstring""" too.
    - Short functions are usually good with short docstrings. A good format for a short docstring (when sufficient) is to state the return directly, like """Return the double of x""" -- for a function called 'twice(x)' -- is clear enough to know what (type) is 'x' and what is returned.

Laboratory: I/O

Create a Python script '`notes.py`' that reads the user input -- interactively, from the terminal -- and write them in to a text file '`notes.txt`'.

To stop the input and "save" the file, the user inputs 'EOF'.

- Each line in 'notes.txt' correspond to each line from the user input;
- When the user inputs the line 'EOF', 'notes.py' closes 'notes.txt'.

Laboratory: I/O

Create a Python script '`notes.py`' that reads the user input -- interactively, from the terminal -- and write them in to a text file '`notes.txt`'. To stop the input and "save" the file, the user inputs 'EOF'.

In '`notes.py`', define a function named '`write_list ( lines_input )`' responsible for writing '`notes.txt`' with content from '`lines_input`'.

- Each line in 'notes.txt' correspond to each line from the user input;
- When the user inputs the line 'EOF', 'notes.py' closes 'notes.txt'.

Laboratory: I/O

Create a Python script '`notes.py`' that reads the user input -- interactively, from the terminal -- and write them in to a text file '`notes.txt`'. In '`notes.py`', define a function named '`write_list ( lines_input )`' responsible for writing '`notes.txt`' with content from '`lines_input`'.

Define another function (in '`notes.py`') -- '`write_read()`' -- that handles the output ('`notes.txt`') and the *input*.

- When the user inputs the line 'EOF', 'notes.py' closes 'notes.txt'.

Laboratory: I/O

Create a Python script '`notes.py`' that reads the user input -- interactively, from the terminal -- and write them in to a text file '`notes.txt`'. In '`notes.py`', define a function named '`write_list ( lines_input )`' responsible for writing '`notes.txt`' with content from '`lines_input`'. Define another function (in '`notes.py`') -- '`write_read()`' -- that handles the output ('`notes.txt`') *and the input*.

Add the possibility for '`notes.py`' to receive the command-line option '-n' which *disables* the writing of user input into '`notes.txt`', instead, just print to *stdout*.

- When the user inputs the line 'EOF', 'notes.py' closes 'notes.txt'.

Laboratory: Exceptions

If your QuestionMark is verifying if each character in the "string is a digit" ([0-9]), the exercise is to modify the solution (QuestionMark) to use '`try/except`' statement to directly operate on those chars as numbers, and *handle* the eventual errors to effectively search for the '?'.

Homework: The Captain's Room

- From Hackerrank
- Define a module as described in our repository: https://github.com/chbrandt/MMM001/blob/master/exercises/assignment_1/the_captains_room/assignment.md
  - With a function '`run()`' with arguments as specified
  - Properly documented
  - With the '`@author: <your name>`' tag

Homework: Monthy-Hall

- Define a module as described in our repository: https://github.com/chbrandt/MMM001/blob/master/exercises/assignment_1/monthy_hall/assignment.md
    - With a function '`run()`' with arguments as specified
    - Properly documented
    - With the '`@author: <your name>`' tag

Assignment: Problems

- Assignment 2 comes with a lot of interesting problems.
  Here we are going to interpret and implement algorithms of good complexity.
- Choose any 3 (three) problems from the 'homework' document in our repository's 'exercise/assignment-2': https://github.com/chbrandt/MMM001/blob/master/exercises/assignment_2/serious_homework.md
- Name the respective Python modules -- implementing the solutions -- accordingly; using underscore '_' to avoid white-spaces and other punctuation/symbols.
- Define a function 'run()' for each module with the solution as stated in the correponding link/platform.