Lecture 1.2: Python Basics

April 5, 2023

Announcements

- Assignment 0 due next Thursday
- Today is the last day to pick a group (more cocktail parties at the end of the day)
  - Once you have a group, sign up for a section on Axess (via assignments > groups on the website)
- Sections start ***next week*** in place of the Thursday lecture (don't come here; instead, go to your section room)
  - Room announcements will be announced on Tuesday in lecture
- We have office hours!

Learning Goals

After today, students will be able to…

- Build a chatbot in Python.
- Define Python terms like “module” and “workspace” and create new modules on their computer.
- Identify resources for determining whether they are using appropriate Python style.

Making a module: `is_prime.py`

Python Files

- You can write and edit code in files. This is the preferred method when you're working on a large codebase or repeatedly editing code.

- Code that should only be executed when the file is being called directly is placed in:

```
if __name__ == '__main__':
    # only executes if this file is being called directly
    ...
```

- Execute the file by calling `python file.py`

Python Style

(a stylish python)

Comments

Comments

```python
# A single-line comment in Python is denoted with the hash symbol.
```

Comments

```python
# A single-line comment in Python is denoted with the hash symbol.

"""
Multi-line comments
Lie between quotation marks
This is a haiku
"""
```

PEP 8

[PEP 8]

Spacing

PEP 8

Spacing

Use four spaces to indent code (don't use tabs).

PEP 8

Spacing

Use four spaces to indent code (don't use tabs).

Use blank lines to separate functions from each other and logical sections within a function.

PEP 8

Spacing

Use four spaces to indent code (don't use tabs).

Use blank lines to separate functions from each other and logical sections within a function.

Use spaces around operators and after commas, but not directly inside delimiters.

```
a = f(1, 2) + g(3, 4) # good
a = f( 1, 2 ) + g( 3, 4 ) # bad
```

PEP 8

PEP 8

Commenting

PEP 8

Commenting

Comment all nontrivial functions.
A function's docstring is the *first (unassigned) string* inside the function body.

PEP 8

Commenting

Comment all nontrivial functions.
A function's docstring is the *first (unassigned) string* inside the function body.

Describe parameters (value / expected type) and return (value / expected type).

PEP 8

Commenting

Comment all nontrivial functions.
A function's docstring is the *first (unassigned) string* inside the function body.

- Describe parameters (value / expected type) and return (value / expected type).
- As usual: list pre/post conditions if any.

PEP 8

Commenting

Comment all nontrivial functions.
A function's docstring is the *first (unassigned) string* inside the function body.

- Describe parameters (value / expected type) and return (value / expected type).
- As usual: list pre/post conditions if any.

Add header comments at the top of files before any imports.

PEP 8

Commenting

Comment all nontrivial functions.
A function's docstring is the *first (unassigned) string* inside the function body.

Describe parameters (value / expected type) and return (value / expected type).

As usual: list pre/post conditions if any.

Add header comments at the top of files before any imports.

If possible, put comments on a line of their own.

```python
def my_function():
    """
    Summary line: do nothing, but document it.

    Longer description: No, really, it doesn't do anything.

    Returns: Gosh, for the last time... nothing (None)!
    """
    pass
```

```python
def my_function():
    """
    Summary line: do nothing, but document it.

    Longer description: No, really, it doesn't do anything.

    Returns: Gosh, for the last time... nothing (None)!
    """
    pass


print(my_function.__doc__)
#    Summary line: do nothing, but document it.
#
#    Longer description: No, really, it doesn't do anything.
#
#    Returns: Gosh, for the last time... nothing (None)!
```

PEP 8

PEP 8

Naming

PEP 8

Naming

Use `snake_case` for variables/functions; `CamelCase` for classes; `CAPS_CASE` for constants.

PEP 8

Naming

Use `snake_case` for variables/functions; `CamelCase` for classes; `CAPS_CASE` for constants.

Decomposition and Logic

PEP 8

Naming

Use `snake_case` for variables/functions; `CamelCase` for classes; `CAPS_CASE` for constants.

Decomposition and Logic

Same as 106A/B/X. Simple is better than complex!

PEP 8

Naming

Use `snake_case` for variables/functions; `CamelCase` for classes; `CAPS_CASE` for constants.

Decomposition and Logic

Same as 106A/B/X. Simple is better than complex!

Automated Code Style Checking

PEP 8

Naming

Use `snake_case` for variables/functions; `CamelCase` for classes; `CAPS_CASE` for constants.

Decomposition and Logic

Same as 106A/B/X. Simple is better than complex!

Automated Code Style Checking

Use PEP8 Online for mechanical violations (naming, spacing) and more advanced suggestions.

PEP 8

Naming

Use `snake_case` for variables/functions; `CamelCase` for classes; `CAPS_CASE` for constants.

Decomposition and Logic

Same as 106A/B/X. Simple is better than complex!

Automated Code Style Checking

Use PEP8 Online for mechanical violations (naming, spacing) and more advanced suggestions.

Use `pycodestyle` as a command line tool. Install with `pip install pycodestyle` (you'll do this in the installation instructions).

Review Activity: Seesaw

- Introduce yourself to your neighbor — well-being inquiries are in order!
- Work through this problem together (website –> lectures –> in class review activity)

For the remaining concepts:

`chatbot.py`

File I/O

```
f = open(filename, method)
```

Read

Function	Action
`next(f)`	Returns the next line in the file
`f.read()`	Returns the entire file as a string
`for line in f:`	Loops over the file, line by line
`f.readlines()`	Returns the lines of the file as a list of strings

Write

Function	Action
`f.write(new_line)`	Writes `new_line` to the file
`f.writelines([collection, of, new, lines])`	Writes the collection of lines to the file

** Writing appends or overwrites, depending on the method*

```
f.close()
```

```
f = open(filename, method)
```

`r`	Read
`w`	Write
`a`	Append
`b`	Bytes Mode

Read

Function	Action
`next(f)`	Returns the next line in the file
`f.read()`	Returns the entire file as a string
`for line in f:`	Loops over the file, line by line
`f.readlines()`	Returns the lines of the file as a list of strings

Write

Function	Action
`f.write(new_line)`	Writes `new_line` to the file
`f.writelines([collection, of, new, lines])`	Writes the collection of lines to the file

** Writing appends or overwrites, depending on the method*

```
f.close()
```

Add a file read loop:

`chatbot.py`

What happens without `f.close()`?

What happens without `f.close()`?

- When the program ends (naturally or from an error), Python will try to clean up any objects that remain in memory.

What happens without `f.close()`?

- When the program ends (naturally or from an error), Python will try to clean up any objects that remain in memory.
- This isn't guaranteed*, but it happens sometimes. You should be concerned if you're writing code that will be run on many operating systems or Python versions.

* Depends on the operating system, Python version, Python implementation (which language the interpreter was written in), ...

What happens without `f.close()`?

- When the program ends (naturally or from an error), Python will try to clean up any objects that remain in memory.
- This isn't guaranteed*, but it happens sometimes. You should be concerned if you're writing code that will be run on many operating systems or Python versions.
- If it isn't closed, the file could remain locked so other programs can't open it or become corrupted.

* Depends on the operating system, Python version, Python implementation (which language the interpreter was written in), ...

What happens without `f.close()`?

- When the program ends (naturally or from an error), Python will try to clean up any objects that remain in memory.
- This isn't guaranteed*, but it happens sometimes. You should be concerned if you're writing code that will be run on many operating systems or Python versions.
- If it isn't closed, the file could remain locked so other programs can't open it or become corrupted.
- The safe option: use a **context manager**!

* Depends on the operating system, Python version, Python implementation (which language the interpreter was written in), ...

```python
with open("words.txt", "r") as f:
```

```python
with open("words.txt", "r") as f:
```

`open("words.txt", "r")` is a file object - it has instructions about how to open and close the file.

The context manager makes sure those instructions are followed, no matter what.

```python
with open("words.txt", "r") as f:
```

`open("words.txt", "r")` is a file object - it has instructions about how to open and close the file.

The context manager makes sure those instructions are followed, no matter what.

```python
with open("words.txt", "r") as f:
```

`open("words.txt", "r")` is a file object - it has instructions about how to open and close the file.

The context manager makes sure those instructions are followed, no matter what.

Roughly equivalent to:

```python
f = open("words.txt", "r")
try:
    ...
finally:
    f.close()
```

Safely read definitions: `chatbot.py`

Strings, Revisited

Useful String Methods

Method	Action
`.lower()`	Converts the string to lowercase
`.upper()`	Converts the string to uppercase
`.title()`	Converts the string to title case (every word capitalized)
`.strip([chars])`	Removes the characters from the ends of the string (or whitespace if chars is omitted)

Method	Action
`.find(substr)`	Finds the first occurrence of `substr` and returns the index (or -1 if not found)
`.replace(old, new)`	Replaces every instance of `old` with `new` and returns the new string
`.startswith(substr)` `.endswith(substr)`	Returns whether the string starts/ends with `substr`

Useful String Methods

Method	Action
`.lower()`	Converts the string to lowercase
`.upper()`	Converts the string to uppercase
`.title()`	Converts the string to title case (every word capitalized)
`.strip([chars])`	Removes the characters from the ends of the string (or whitespace if chars is omitted)

Method	Action
`.find(substr)`	Finds the first occurrence of `substr` and returns the index (or -1 if not found)
`.replace(old, new)`	Replaces every instance of `old` with `new` and returns the new string
`.startswith(substr)` `.endswith(substr)`	Returns whether the string starts/ends with `substr`

Splitting and Joining

```
"3-14-2015".split('-')
```

Splitting and Joining

```
"3-14-2015".split('-')
```

Splitting and Joining

```
"3-14-2015".split('-')  # => ['3', '14', '2015']
```

Splitting and Joining

```
"3-14-2015".split('-')  # => ['3', '14', '2015']

"Tara Elizabeth Jones".split()
```

Splitting and Joining

```
"3-14-2015".split('-')  # => ['3', '14', '2015']

"Tara Elizabeth Jones".split()
```

Splitting and Joining

```
"3-14-2015".split('-')  # => ['3', '14', '2015']

"Tara Elizabeth Jones".split()
# => ['Tara', 'Elizabeth', 'Jones']
```

Splitting and Joining

```
"3-14-2015".split('-')  # => ['3', '14', '2015']

"Tara Elizabeth Jones".split()
# => ['Tara', 'Elizabeth', 'Jones']

", ".join(["Arpit", "Chase", "Will"])
```

Splitting and Joining

```
"3-14-2015".split('-')  # => ['3', '14', '2015']

"Tara Elizabeth Jones".split()
# => ['Tara', 'Elizabeth', 'Jones']

", ".join(["Arpit", "Chase", "Will"])
# => 'Arpit, Chase, Will'
```

Finish the chatbot:

`chatbot.py`