Introduction to Computation for the Humanities and Social Sciences

CS 3

Chris Tanner

HOW'S THE DEBUGGING GOING?
JORGE CHAM © 2005
www.phdcomics.com

Lecture 5

“But I’ve got a blank space
[in my text String] baby”
— Taylor Swift

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Reading Input

- Programs need data/input in order to operate
- Input can come from:
    - someone using your program
    - external source (e.g., files on your computer, live web page like Twitter, etc)

Reading Input from a User

```
name = input(“What is your name? “)
age_string = input(“What is your age? “)
```

- A **function** that collects input from the command-line terminal, and returns (aka outputs) the input value as a text string
- Useful when used in a program (.py), not in Python’s interactive mode
- The input argument prompt will be displayed to the user on the same line as their input, so it’s helpful to include some whitespace at the end of your prompt so the user can easily see the start of their input

Reading Input from a User

Everything returned by **`input()`** will be a text string, so if you're actually wanting text data, that's fine. Otherwise, if you want to store the text input as a numerical data type, we need to **cast** it accordingly.

```
age_string = input(“How old are you? “)
age = int(age_string)
```

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

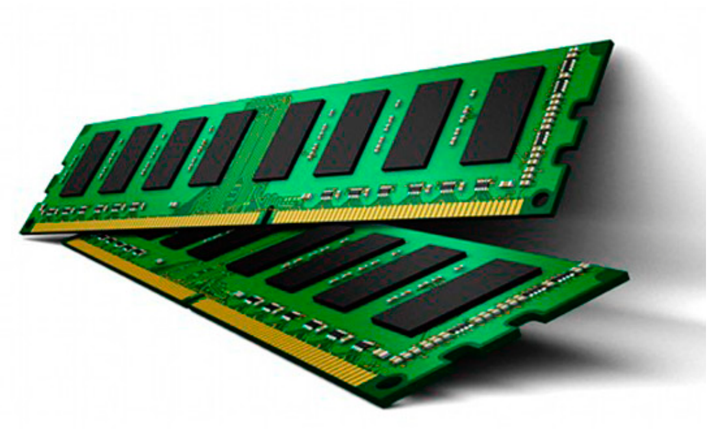
Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Reading Input

Computer Storage

VS



Large, slow, permanent

Small, fast, temporary

Reading Input

The input() function is fairly impractical

```
script_text = input(“Type in the script for \
      The Office - Season 5, Episode 7: ”)
```

Reading Input from a File

Opening/Closing file

- With our programs so far, we have been storing data to variables. This occurs in our computer's memory (which is temporary)
- The computer does not remember this information after your program is run
- Your computer can also permanently store information on its hard drive
- Saving your program's file (.py code) is an example of permanently writing your file to the hard drive, as is any other document you write
- Reading and writing files from/to the hard drive represents another input/output method for our programs

Reading Input from a File

Opening/Closing file

- For both reading and writing, you need to write Python code to “open” the file
- After you are done reading/writing, your program needs to “close” it
- When writing to a file, if you don’t close your file, ~~your computer will explode~~ the changes you made might not be saved
- After closing the file, you can’t read or write from it again without opening it again

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
```

Reading Input from a File

Example

```
input_file = open("example.txt", 'r')
```

the **filename** of the actual file that already exists on your computer, which we'll read.

specifies to open the file for **reading** purposes.

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
```

the contents of the file are now temporarily stored/accessible via this variable we’re creating

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
```

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
for line in input_file:
    print(line)
input_file.close()
```

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
for line in input_file:
    print(line)
input_file.close()
```

must be 4 spaces

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
for line in input_file:
    print(line)
input_file.close()
```

NOTE: the end of every line you read will contain a special character (i.e., pressing ENTER) which represents “end of line, go to the next line.” To remove this extra spacing, we need to call the function **.strip()**

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
for line in input_file:
    line = line.strip()
    print(line)
input_file.close()
```

Reading Input from a File

Example

- It's practice to run `.strip()` on every line from an input file
- If the file is delimited by a certain character, we can **split()** the line up into each of its meaningful pieces (i.e., columns of data)
  - some files are delimited by a **comma** (.csv files), which denotes each piece of data from the other
  - other times, just separating a line by its spaces could be highly useful — e.g., just loading every word from a line!

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
for line in input_file:
    name, age, weight = line.strip().split(“,”)
    print(weight)
input_file.close()
```

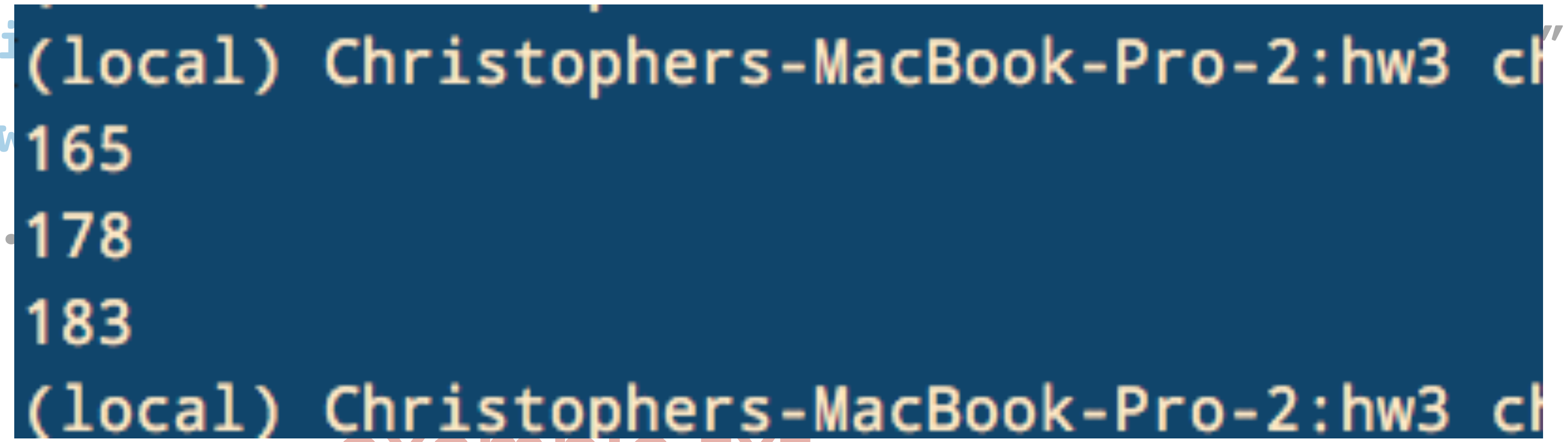
input: example.txt

```
Michael Scott,47,165

Jim Halpert,33,178

Dwight Shrute,36,183
```

Reading Input from a File

Example

```
input_file = open(“example.txt”, ‘r’)
for line in input_file:
    name, i                                          ”)
    print(w
input_file.
```



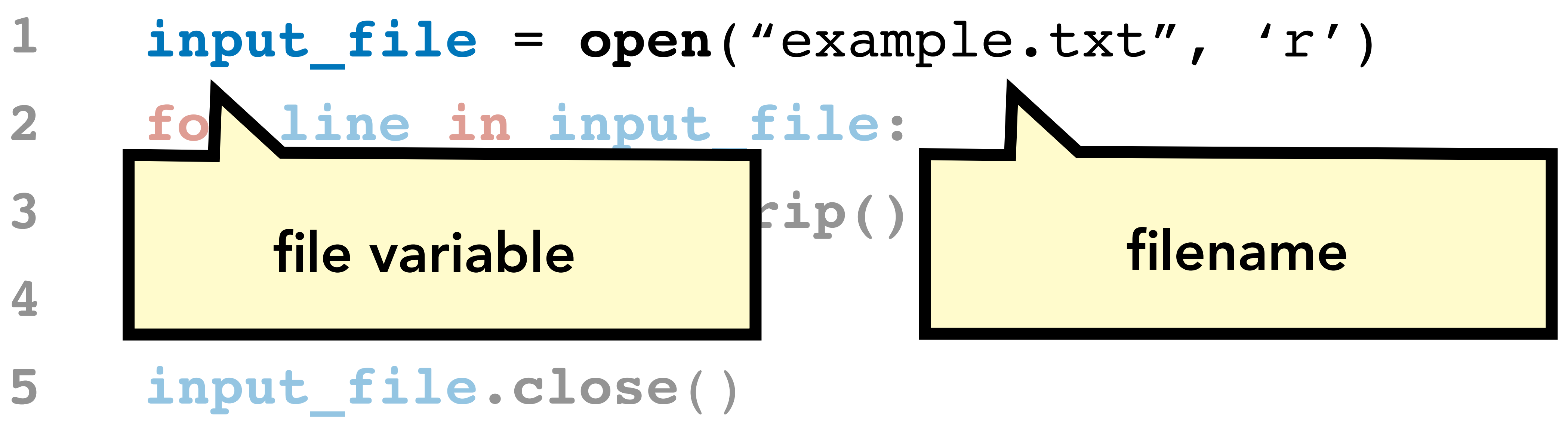
```
(local) Christophers-MacBook-Pro-2:hw3 ch
165
178
183
(local) Christophers-MacBook-Pro-2:hw3 ch
```

example.txt

```
Michael Scott,47,165
Jim Halpert,33,178
Dwight Shrute,36,183
```

Reading Input from a File

Filename vs File



```
input_file = open(“example.txt”, ‘r’)
fo  line in input file:
                   rip()

input_file.close()
```

- The **filename** is just a string that represents the **path to the file** from the location you ran your program from
- The **file** variable connects the program to the file on the computer and its contents

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Writing to a File

Opening a File to Read:

```
input_file = open(“example.txt”, ‘r’)
```

specifies it’s for **writing**

Opening a File to Write:

```
output_file = open(“example.txt”, ‘w’)
```

Writing to a File

Opening a File to Read:

```
input_file = open(“example.txt”, ‘r’)
```

Opening a File to Write:

```
output_file = open(“example.txt”, ‘w’)
output_file.write(“Hello!”)
output_file.close()
```

Writing to a File

Opening a File to Read:

```
input_file = open(“example.txt”, ‘r’)
```

Opening a File to Write:

```
my_name = “Chris”
output_file = open(“example.txt”, ‘w’)
output_file.write(“Hello, my name is:” + my_name)
output_file.close()
```

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Errors

We've seen three specific errors so far (in Lecture 4):
NameError, **TypeError**, and **ValueError**

In general, there are **three categories** of errors:

- syntax errors (includes TypeError)
- runtime errors (includes NameError and ValueError)
- logic errors

Syntax Errors

- If your code is improperly formed, a **syntax error** will occur during the compilation stage (before your code is executed, and when your code is being translated to lower-level *byte-code*).
- They show up immediately, no matter where the issue exists in the code
- These are basically grammar issues, where you're doing something that Python is smart enough to know ahead of time is wonky.
- The easiest of the errors to fix.

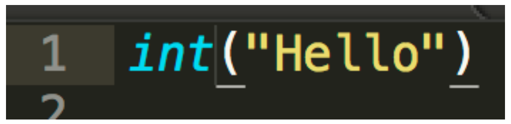
Runtime Errors

- A **runtime error** occurs when your code is actually being executed, and Python can't perform what you instructed it to do.
- Very common. Anytime you're trying to access something incorrectly according to how you've defined your variables/data.
- Examples include mistyped variable names, using a variable before it's defined, etc.

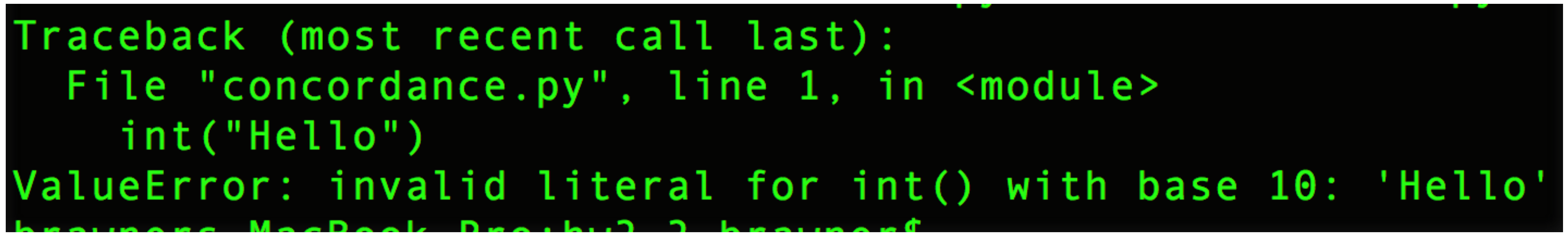
Errors

Runtime Errors

- An **exception** is **thrown** whenever the interpreter reaches a line of code that results in an improper calculation
- These exceptions occur at the specific point in time the improper calculation is encountered
- For a long running program, these may occur late in your program



```python
int("Hello")
```



```
Traceback (most recent call last):
  File "concordance.py", line 1, in <module>
    int("Hello")
ValueError: invalid literal for int() with base 10: 'Hello'
```

Logic Errors

- **Logic Errors** is a general term used to describe when your code executes but gives different behavior/results than what you intended.
- May or may not yield a runtime error.
- It's easy to define computations incorrectly, so start small, check your intermediate values often, and be methodical and patient (per Lecture 2)

Error Handling

- If you anticipate that some data may differ from how you intend, you can write code to appropriately/gracefully respond, instead of just crashing.
- Use **`Try`** to represent the main action that ideally want to execute
- Use **`Except`** to anticipate a potential error type that would occur if used improperly
- Within the Exception, do whatever you'd like in terms of recovering/ fixing the error or just informing the user and optionally exiting the program.

Type Checking

```
input_file = open(“example.txt”, ‘r’)
for line in input_file:
    name, age, weight = line.strip().split(“,”)
    try:
        weight = int(weight)
    except TypeError:
        print(“ERROR: Please enter an integer \
        (whole number) for weight!”)
        exit(1)

    print(name + “ weighs “ + str(weight) + “ pounds”)
input_file.close()
```

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Lecture 5

- Reading Input From Users
- Reading Input From Files
- Writing Data to Files
- Errors
- Style Guide

Style Guide

- **Having correct code is the goal and most important element of programming**
- However, the exact words you type (e.g., names you give to variables) also have some rules to follow — some of which are mandatory by Python, others are just generally good practices for ensuring readable, nice-looking code.

Python's Naming Rules for Variables

- Variable names must contain only the characters a-z, A-Z, 0-9 and underscores

✓ `weight_per_pound`
✓ `result`
✗ `help!`
✗ `don't have a cow`

- A letter or underscore must be the first character

✓ `result_2`
✓ `boys_2_men`
✗ `2nd_result`
✗ `98_degrees`

- Variable names are case sensitive

`result` is not same as `Result`

Variable Names Allude to Type

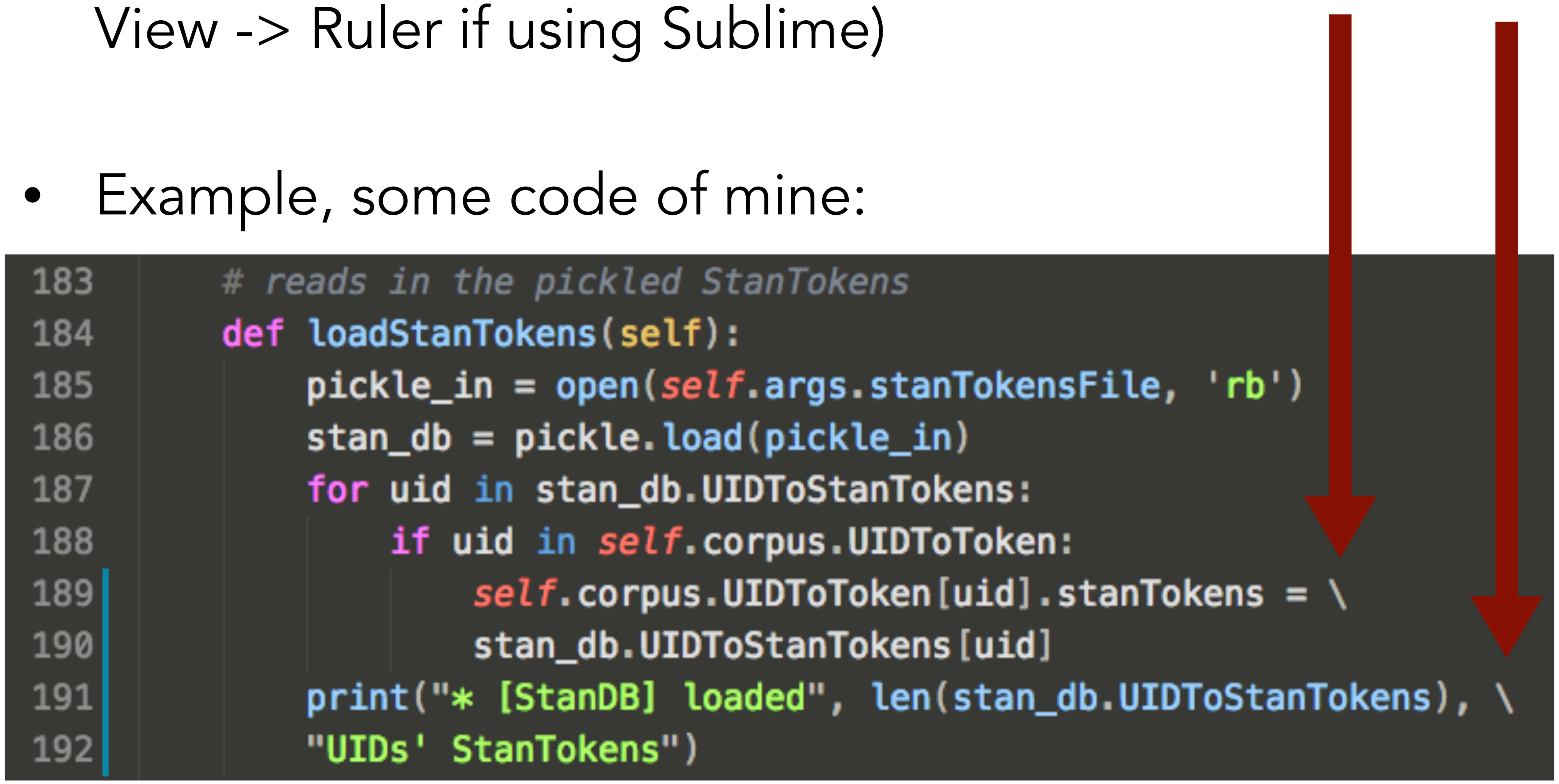
- **`int`**s: num_employees, paper_count, list_index, etc
- **`float`**s: temperature, employee_weight, miles_travelled, etc
- **`string`**s: filename, book_title, word, sentence, book_description, employee_name, etc
- **`boolean`**s: is_available, is_sorted, is_full, etc

Style Guide

Python's Style Rules

- If a line of code extends beyond 80 characters long, break it up onto another line via wrapping parts in "(" and ")" or a "\" at the end of the long line View -> Ruler if using Sublime)

- Example, some code of mine:



```python
    # reads in the pickled StanTokens
    def loadStanTokens(self):
        pickle_in = open(self.args.stanTokensFile, 'rb')
        stan_db = pickle.load(pickle_in)
        for uid in stan_db.UIDToStanTokens:
            if uid in self.corpus.UIDToToken:
                self.corpus.UIDToToken[uid].stanTokens = \
                stan_db.UIDToStanTokens[uid]
        print("* [StanDB] loaded", len(stan_db.UIDToStanTokens), \
        "UIDs' StanTokens")
```

Python’s Style Rules

- Later, much of our code will need to be indented (as we saw for File reading/writing and lines that are > 80 characters)
- Always indent by typing 4 spaces (not TAB).
- Or, if you use TABS, make sure you tell your IDE (e.g., Atom, Sublime) to “convert TABS to SPACES)

Python’s Style Rules

For a comprehensive set of the Python’s Style Guide (which helps answers nuanced cases), please visit the PEP 8 Style Guide

Style Guide

Comments

- **very useful:** code can be difficult to read, not only for others but even to yourself later on

- to make it more readable, we can write some typical English comments/notes in our code, wherever we want!

- single-lined comments must be prefaced with #

- # tells Python to ignore that line, as it's not Python code

- if you have several lines of comments, you can just surround the text with ''' before and after

Comments

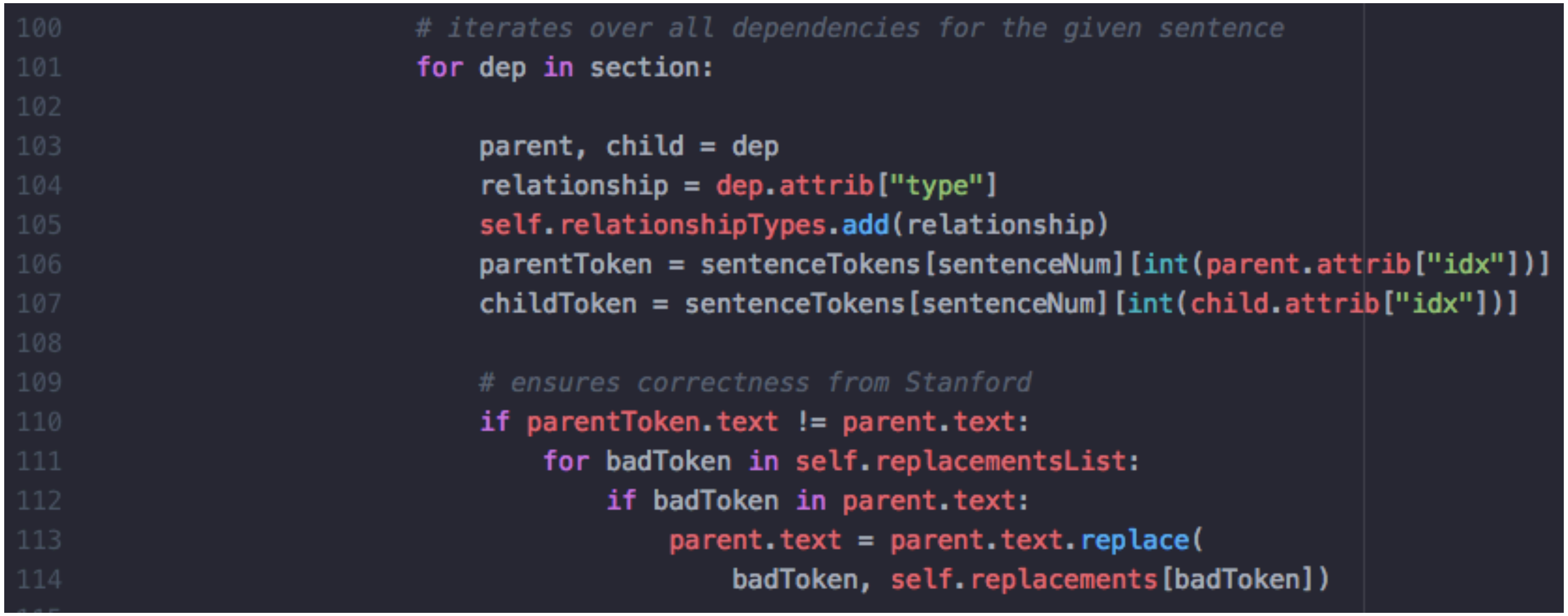
Examples:

```
# initializes variables
my_age = 18
my_name = "Charlie"

'''
looks at votes from all senators over
the past 10 years, and for each year
displays the top 5 senators who voted
most dissimilar from the rest of their political party
'''
def loadListOfSenators(file):
    ... code here ...
```

Style Guide

Comments

- It's recommended to comment main sections of code
- Commenting too much is ridiculous and distracting
- Here's a healthy amount:



```
            # iterates over all dependencies for the given sentence
            for dep in section:

                parent, child = dep
                relationship = dep.attrib["type"]
                self.relationshipTypes.add(relationship)
                parentToken = sentenceTokens[sentenceNum][int(parent.attrib["idx"])]
                childToken = sentenceTokens[sentenceNum][int(child.attrib["idx"])]

                # ensures correctness from Stanford
                if parentToken.text != parent.text:
                    for badToken in self.replacementsList:
                        if badToken in parent.text:
                            parent.text = parent.text.replace(
                                badToken, self.replacements[badToken])
```

Style Guide

Our Style Rules

- Only use lowercase characters
- Separate words with underscores
- Variable names should contain full words or common abbreviations (e.g., avg)
- Don’t make up your own abbreviations or shorthand (e.g. s for sum/string)
- Don't use a reserved Python keyword (any word not in quotes that’s colored in your text editor)
    - Including a Python word as part of the variable name is OK (weight_string, user_input)
- Variable names should include a word that alludes to its type

Style Guide

Name That Variable

- The number of senators in a file
- The average age of a group of people
- The name of the user of our program, per their input
- A date of birth for someone
- The name of a city about which we will analyze data
- The total number of people affected by a power outage

Lab Time