Lists, Data, Plotting

6.1000 LECTURE 4

Announcements

- Pset 1 due Wednesday 9/17
- Tomorrow is last day to switch to 6.100A
- Viruses are spreading – stay healthy!
  - prioritize sleep and water
  - eat enough
    - get fruits and vitamin C
  - schedule your flu shot

Loose ends: Syntax

- Triple-quoted strings
- Statements vs expressions
  - **statements** are any instruction to Python
  - **expressions** are statements that evaluate to an object
  - non-expression statements
    - `if-elif-else`
    - `while, for, continue, break`
    - `def ..., return ...`
    - many more
      - https://docs.python.org/3/reference/simple_stmts.html
      - https://docs.python.org/3/reference/compound_stmts.html

Loose ends: Terminology

- **Generate-and-test / guess-and-check**
  - very broad term for any algorithm that checks candidates along the way to a final answer
  - **exhaustive enumeration / brute force**
    - systematically considers every possibility in the solution space
    - expected to be slow
  - **pruning**
    - discards significant chunks of the solution space through inference
    - **bisection search** is an example

Loose ends: Terminology

- Function names
  - `def weirdo(achoo):`
    `    print(achoo + achoo)`
  - `weirdo` is the function name
    - evaluates to the function object
  - `weirdo("honk")` is a function call
    - evaluates to the function's return value, None
  - often use `weirdo()` to refer to the function itself
    - official Python documentation does this
    - helps clarify that we're talking about a function

Loose ends: Frames and function scope

- **Frame** is simply a table from variable names to locations of objects in memory
  - **global frame,** function call **local frames**
  - frames live in the **stack** section of memory
  - objects live in the **heap** section
  - **frames do not contain objects, only references to them** (i.e., memory addresses)
- **What happens when function body references a variable not in function's local frame?**
  - Python automatically looks up the variable in the global frame

Lists

- **Sequence collections of objects**
- Not strictly necessary for computation
  - Turing machines have no lists
- But awfully convenient for grouping data, e.g.,
  - get sequence of input values, but don't know in advance how many
  - return more than one piece of data from a function

Lists: Literals

- Square brackets around comma-separated values
  - `[1, 3, 5, 7, 9, 11]`
- Lists are just sequences of references to other objects
  - **each cell is like a variable**
  - **objects do not live inside lists!**
  - nested lists are really list cells pointing to other list objects

Lists: Operations

- As a sequence, very similar operations to **`str`**s
  - indexing, slicing
  - concatenation, repetition
  - looping, `len()`
- Differences to note
  - **`str`**s contain characters, but no actual data contained in a **`list`**
  - hence indexing and looping do not create new objects, but just reference what the list cells already point to
  - slicing still always creates a new list
- Exercise: study the similarities and differences between **`str.find()`** and **`list.index()`**
  - https://docs.python.org/3/library/stdtypes.html#sequence.index
  - https://docs.python.org/3/library/stdtypes.html#str.find

Lists: Comparisons

- Also similar to **`str`** comparisons
  - uses **==** and **<** on elements
- But **nesting** means checks may involve more structure
  - comparing two lists means using same comparison operator on pairwise elements
  - `listA == listB`
    - `listA[0] == listB[0]`
      `and listA[1] == listB[1]`
      `and listA[2] == listB[2]`
    - can stop at first pair comparison that's `False`

Lists: Mutation

- All previous object types were **immutable**
  - once object is created, its contents won't change
  - **list objects are designed to be changed**
- Changing the references of list cells
  - **list cells are just like variables**
  - they can be "re-assigned" to any other object at any time
- Changing how many cells there are
  - `append()` and `extend()` put new cells at end
  - `del list_obj[index]` removes reference at `index`
    - shifts all subsequent cell references one index earlier
- Will revisit list mutation next Monday in Lecture 6

Functions operating on lists

- `list()` makes a new list from elements of an iterable into
  - `a = list(iterable)` is equivalent to

```
a = []
for elt in iterable:
    a.append(elt)
```

  - iterables we know so far are **`str`**, **`range`**, **`list`**
- `str.split()` and `str.join()`
  - https://docs.python.org/3/library/stdtypes.html#str.split
  - https://docs.python.org/3/library/stdtypes.html#str.join
- `sorted()` take in an iterable, always produces a list
  - https://docs.python.org/3/library/functions.html#sorted

Reading files

- `open()` returns a file object for reading
  - actual file lives on disk
    - files are sequences of bytes (1 byte is a chunk of 8 bits)
  - `open(filename)` creates an object that represents access to that file
    - Python takes care of low-level details
- getting file contents
  - `file.read()` returns all content as a `str`
    - typically one byte becomes one character
  - `file.readline()` gets all content until next newline `"\n"`
  - `file.readlines()` returns a list of all lines
  - can iterate over lines in file directly; these are equivalent:
    - `for line in file:`
    - `for line in file.readlines():`

Closing files

- When no longer need file, good practice to call `file.close()`
  - an open file means Python is still ready to read or write to it
  - on some operating systems, prevents other programs from accessing the file
- Recommend using `with open(filename) as file:` statements
  - automatically closes files when block ends
  - uses a Python feature called context managers

Plotting data: Installing matplotlib

- **matplotlib** is a popular third-party library for generating plots
  - https://matplotlib.org/stable/
- Need to add to your Python installation
  - Windows
    ```
    > py -m pip install matplotlib
    ```
  - macOS
    ```
    % python3 -m pip install matplotlib
    ```
- **pip** = Package Installer for Python
  - pip is a built-in module
  - the install command retrieves matplotlib code and makes it an available module as well
- **modules** are collections of names pointing to useful objects
  - kind of like frames, but they live in the **heap** where all other objects live
  - **`import matplotlib`**
    - loads library of functions
    - makes module object, creates **`matplotlib`** variable pointing to it

Plotting data: Using matplotlib

- `import matplotlib.pyplot as plt`
  - `pyplot` is a submodule supplying matplotlibs' primary user-interface
  - `plt` is a convenient name we put in the global frame referencing it
- **make a new figure**
  - `plt.figure()`
- **draw on the figure**
  - draw data: `plt.bar(), plt.plot()`
  - label the graph: `plt.title(), plt.xlabel(), plt.grid()`
  - customize colors: `plt.plot(x, y, marker=..., color=...)`
- **display the figure**
  - `plt.show()`

https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.bar.html
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html
https://matplotlib.org/stable/users/explain/colors/colors.html
https://matplotlib.org/stable/api/markers_api.html