This document: **http://bit.ly/resbaz-python**

Helpful Links :)

- **Syllabus** at: https://denubis.github.io/2019-09-10-RezbazSyd-IntroPython/
- **Python Lesson**: http://swcarpentry.github.io/python-novice-inflammation/
    - Python setup: http://swcarpentry.github.io/python-novice-inflammation/setup/
    - Data: http://swcarpentry.github.io/python-novice-inflammation/data/python-novice-inflammation-data.zip
    - Code: http://swcarpentry.github.io/python-novice-inflammation/code/python-novice-inflammation-code.zip
- **References (Cheat-sheets)**
    - Python: http://swcarpentry.github.io/python-novice-inflammation/reference/

Rules of Debugging

1. Fail early, fail often.
2. Always initialize from data.
3. Know what it's supposed to do.
4. Make it fail every time.
5. Make it fail fast.
6. Change one thing at a time, for a reason.
7. Keep track of what we've done.
8. Be humble.
9. Test the simple things first.
10. Talk to someone, even if it is a rubber duck.

And remember, a week of hard work can sometimes save you an hour of thought.

Live space for Questions or Comments on the day

Anyone may write here if they have questions or comments on the lesson. Instructors will be monitoring this document as well.

-

Instructors

Brian Ballsun-Stanton (Lead)

Brian Ballsun-Stanton is Solutions Architect (Digital Humanities) for the Macquarie University Faculty of Arts with a PhD from UNSW in Philosophy. He is working with researchers from across Australia to deploy digital technologies. Brian's current research interests are in exploring the Philosophy of Science's interactions with the Open Data movement, and building tools for rapid analysis and bulk manipulation of large ancient history corpora.

Conrad Gillard (Co-Instructor)

Conrad Gillard is a PhD student at the University of New South Wales, in the school of chemistry. Conrad's area of research is on batteries and superconductors. Conrad uses Python and other programming languages for his work, and hopes to help others learn how to use these.

c.gillard@unsw.edu.au

Fiona Jones (Helper)

Fiona Jones is a Research Librarian, supporting the research, teaching and learning activities of the Faculty of Science and Engineering. She is interested in answering these questions: How can research and data skills development in research students be supported by librarians? How can librarians develop their own skills and build capacity for supporting the university community?

Wilfred Gee (Helper)

Wilfred Gee is a PhD candidate in the Department of Physics and Astronomy Faculty of Science working on detecting extrasolar planets in data obtained from an open source citizen science program. Wilfred has a background as a full-stack web applications developer but currently spends time on hardware control and data analysis. Most of Wilfred's work is done in the Google Cloud Environment so he also has experience building and managing pipeline infrastructure in the cloud.

Nishen Naidoo (Helper)

Nishen is a Senior Systems Analyst for Macquarie University Library specialising in systems integrations and interoperability. Nishen has a passion for learning new technologies and identifying how best they can be deployed to solve current (or future) problems. Recently his interests have been geared towards machine learning and big data analysis and the tools and technologies that are used to support these.

Lei Han (Helper)

Lei Han is a post-doc research scientist at Macquarie University Library specialising in business process management and data analysis. Lei's research interests include business process management and improvement, and process mining from large datasets and process model analysis.

John Zaitseff (Helper)

John Zaitseff is a Research Computing Support Engineer at UNSW Sydney. He provides both wide and deep expertise in all aspects of High Performance and Research Computing to researchers, including analysing computing needs and requirements, providing advice on purchasing and utilisation, training staff and students to use new and existing facilities, and performing software design, programming, debugging and code optimisation using multiple programming languages.

Mark Minchenko (Helper)

Mark Minchenko is a Research Computing Support Officer at UNSW Sydney. He assists in managing High Performance and Research Computing Resources as well as providing user support for these resources.

Code of Conduct

https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

To summarise:

All participants in our events and communications are expected to show respect and courtesy to others. All interactions should be professional regardless of platform: either online or in-person. In order to foster a positive and professional learning environment we encourage the following kinds of behaviours in all Carpentries events and platforms:

- Use welcoming and inclusive language
- Be respectful of different viewpoints and experiences
- Gracefully accept constructive criticism
- Focus on what is best for the community
- Show courtesy and respect towards other community members

Event Wi-Fi Setup

Eduroam

use your home institution credentials - test before you come at your own home institution

UNSW WiFi

Pre-register or use your phone to register at: https://bit.ly/resbaz-wifi
Event Passphrase: resbaz2019!

SSID: `uniwide`
Username: issued by registration process above
Password: issued by registration process above

Schedule

- Day 1 - Tuesday - Stream 1
    - Before Pre-workshop survey
    - 10:00 - 12:00 Analysing Patient Data with Python
    - 12:00 - 13:00 Resbaz Festival and Lunch
    - 13:00 - 15:00 Repeating Actions with Loops in Python
        - Storing Multiple Values in Lists
- Day 2 - Wednesday - Stream 1
    - 10:00 - 12:00 Analysing Data from Multiple Files with Python
        - Making Choices with Python
    - 12:00 - 13:00 Resbaz Festival and Lunch
    - 13:00 - 15:00 Creating Functions in Python

Welcome to The Carpentries Shared Workspace!

This pad is synchronized as you type, so that everyone viewing this page sees the same text. This allows you to collaborate seamlessly on documents.

Use of this service is restricted to members of The Carpentries community; this is not for general purpose use (for that, try Shared Workspace.wikimedia.org).

Users are expected to follow our code of conduct: https://docs.carpentries.org/topic_folders/policies/code-of-conduct.html

---

Sign in: Name, Institution, Email, Twitter (optional)

Please sign in so we can record your attendance.Paula Ngu, UNSW, mkpaula8@hotmail.com

- Julian Peters, UNSW, julian.peters@unsw.edu.au
- Adam Theobald, Black Dog Institute; a.theobald@blackdog.org.au
- Daria Kalmanovich,UNSW, d.kalmanovich@unsw.edu.au
- Shah Mahdi Hasan, University of Newcastle, shahmahdi.hasan@uon.edu.au
- Paula Ngu, UNSW, mkpaula8@hotmail.com
- Gerald Schwert-Strachwitz, MGSM, gerald.schwert-strachwitz@students.mq.edu.au
- David Simmons, University of Sydney, dsim0140@sydney.edu.au
- Sandrine Chan, Macquarie University, sandrine.chanmoifat@hdr.mq.edu.au
- Rodrigo Araujo, Macquarie University; rodrigo.araujo-e-castro@students.mq.edu.au
- Megan Daniels, University of New England; megan.daniels@une.edu.au
- Omid Ghasemi, Macquarie University
- , omidreza.ghasemi21@gmail.com
- Fei Ji, NSW Department of Planning, Industry and Environment, fei.ji@environment.nsw.gov.au
- Shyno Susan John, University of Wollongong,ssj703@uowmail.edu.au
- Eline Smit, Western Sydney University, e.smit@westernsydney.edu.au
- Daniel Hochstrasser, Western Sydney University,, d.hochstrasser@westernsydney.edu.au
- Brenda Vo, University of New England bvo3@une.edu.au
- Kelvin Lee, University of Sydney, kelvin.k.lee@sydney.edu.au
- Emi Iwasaki, Macquarie University, emi.iwasaki@hdr.mq.edu.au
- Maria Mempin, Macquarie University, maria.mempin@mq.edu.au
- Suneha Seetahul, University of Sydney, suneha.seetahul@sydney.edu.au
- Abhay Kulkarni, Macquarie University, Applied Finance, abhay.kulkarni@hdr.mq.edu.au
- Charuni Pathmeswaran, UNSW, c.pathmeswaran@student.unsw.edu.au, @Charuni_

- Mia Gross, Department of Planning, Industry and Environment, mia.gross@environment.nsw.gov.au
- Tahereh Jalalabadi, UON , tahereh.jalalabadi@uon.edu.au
- Wilfred Gee - Macquarie University, wilfred.gee@mq.edu.au
- Katherine Jackson, Garvan Institute of Medical Research, k.jackson@garvan.org.au
- Jessica Houang, USYD, jessica.houang@sydney.edu.au
- Vanessa Crosby, UNSW, @alienspectacles
- Parice Brandies, University of Sydney, parice.brandies@sydney.edu.au, @PariceBrandies
- Nishen Naidoo, Macquarie University, @nish_naidoo
- Marilia Menezes de Oliveira, University of Sydney, marilia.menezes@sydney.edu.au
- Lauren Castle, UNSW, l.castle@unsw.edu.au
- Amalia Halim, UNSW, a.halim@unsw.edu.au
- Deborah Apthorp, University of New England, dapthorp@une.edu.au
- Amanda, UNSW, yun.luo1@student.unsw.edu.au
- Neil Orr, University of Sydney, neil.orr@sydney.edu.au
- Tahereh Tekieh, University of Sydney, tahereh.tekieh@sydney.edu.au

Please make sure you have filled out the pre-training survey.

Icebreaker

- Introduce yourselves to your table.
- Tell each other what you want to get out of the day.
- One member from each table will provide an example to the room.
- https://imgs.xkcd.com/comics/python.png

Analyzing Patient Data

https://denubis.github.io/python-novice-inflammation/01-numpy/index.html

Questions:

- How can I process tabular data files in Python?

Objectives:

- Explain what a library is and what libraries are used for.
- Import a Python library and use the functions it contains.
- Read tabular data from a file into a program.
- Assign values to variables.
- Select individual values and subsections from data.
- Perform operations on arrays of data.
- Plot simple graphs from data.

Topics covered in Exercise 1

Navigation of Jupyter Notebook Environment

- Windows ‘Start menu’ - anaconda prompt - type ‘Jupyter notebook’ - click on ‘New’ - Python 3 - new window - click on ‘untitled’ in the top left corner - Rename
- **Note**: You can use Markdown chunks in Jupyter notebooks - use the menu at the top.
- Accessing help can be done by simply using “?” after the function. For example: print?. For user defined functions, you can see the source code by using ‘??’ after the function name.
- Access help / explanations by holding down Shift and pressing Tab once. Press Tab twice for more verbose explanations. Or press tab three or four times to open the help separately
- You can access and edit your saved Notebook directly from Jupyter homepage.
- Use # in Jypyter notebook to insert a comment

Variables

- Assign numbers/outputs to variables to use them later. When you type the variable then hit “shift enter” you’ll see the value of that variable. Alternatively use print(variable name).
- Note: variables are case sensitive.
- Helpful to think of variable as sticky-notes

Types of data

- type (variable_name)

- String
- # gives you the type of your variable e.g. str (string) int (integer), etc.

Functions

- Functions can use more than one argument e.g. print(name, “is my name”)
- Same concept to y=f(x)
- To display the help for a function: shit+tab or shift+tab+tab or shift+tab+tab+tab

Using Variables

- Important to remember is that variables are not updated automatically
- Code is executed **sequentially.**

Callout: Variables as Sticky Notes

A variable is analogous to a sticky note with a name written on it: assigning a value to a variable is like putting that sticky note on a particular value.

Image: Variables as Sticky Notes https://denubis.github.io/python-novice-inflammation/fig/python-sticky-note-variables-01.svg

This means that assigning a value to one variable does **not** change values of other variables. For example, let’s store the subject’s weight in pounds in its own variable:

Python:

```
# There are 2.2 pounds per kilogram
weight_lb = 2.2 * weight_kg
print(weight_kg_text, weight_kg, 'and in pounds:', weight_lb)
```

Output:

```
weight in kilograms: 65.0 and in pounds: 143.0
```

Image: Creating Another Variable https://denubis.github.io/python-novice-inflammation/fig/python-sticky-note-variables-02.svg

Let's now change `weight_kg`:

Python:

```
weight_kg = 100.0
print('weight in kilograms is now:', weight_kg, 'and weight in pounds is still:', weight_lb)
```

Output:

```
weight in kilograms is now: 100.0 and weight in pounds is still: 143.0
```

Image: Updating a Variable https://denubis.github.io/python-novice-inflammation/fig/python-sticky-note-variables-03.svg

Since `weight_lb` doesn't "remember" where its value comes from, it is not updated when we change `weight_kg`.

Loading data into Python

Topics covered in Exercise 2

Importing Libraries

- To import a library use the function 'import' + name of library (e.g. import numpy)
- Intelligent code completion - environment assists by suggesting endings to code lines
- NumPy is a commonly used data manipulation library for python

Loading data

- numpy.loadtext(fname = "path", delimiter = ',')
- Type "r" in front of filepath with \ symbols
- numpy.loadtxt(fname=r'Desktop/data/inflammation-01.csv', delimiter=',')
- Google errors (StackExchange)

Listing possible functions and arguments

- Properties versus methods

Array operations

- In a range (e.g. data[1:4,3:6], last number is not inclusive
- **Reminder:** when you call specific data from an array using the 'slice' method (e.g. 1:4) it will only output up to (and not including) the final digit (in this case, 4). this is simply a quirk of the Python language.
- When calling an element in an array, row first, column second!

- The array count starts from 0, 1, 2…
- When calling all rows or all columns, use :

Generating Plots

- Matplotlib.pyplot is library to generate plots
- matplotlib.pyplot.show() to show plots
- matplotlib.pyplot.figure()
- subplot1 = fig.add_subplot(1,3,1) # Note: Number refer to position,
- # number of rows, number of columns
- subplot1.set_ylabel('title')

Callout: Mystery Functions in IPython

How did we know what functions NumPy has and how to use them? If you are working in IPython or in a Jupyter Notebook, there is an easy way to find out. Firstly, if you are in Jupyter Notebook, insert an additional cell at the top of your notebook and enter the following code: *%config IPCompleter.greedy=True*. Then, if you type the name of something followed by a dot, you can use tab completion (e.g. type `numpy.` and then press tab) to see a list of all functions and attributes that you can use. After selecting one, you can also add a question mark (e.g. `numpy.cumprod?`), and IPython will return an explanation of the method! This is the same as doing `help(numpy.cumprod)`.

Callout: Importing libraries with shortcuts

In this lesson we use the `import numpy` syntax to import NumPy. However, shortcuts such as `import numpy as np` are frequently used. Importing NumPy this way means that after the initial import, rather than writing `numpy.loadtxt(...)`, you can now write `np.loadtxt(...)`. Some people prefer this as it is quicker to type and results in shorter lines of code - especially for libraries with long names! You will frequently see Python code online using a NumPy function with `np`, and it's because they've used this shortcut. It makes no difference which approach you choose to take, but you must be consistent as if you use `import numpy as np` then `numpy.loadtxt(...)` will not work, and you must use `np.loadtxt(...)` instead. Because of this, when working with other people it is important you agree on how libraries are imported.

Callout: Data Type

A Numpy array contains one or more elements of the same type. The `type` function will only tell you that a variable is a NumPy array but won't tell you the type of thing inside the array. We can find out the type of the data contained in the NumPy array.

Python:

```
print(data.dtype)
```

Output:

```
float64
```

This tells us that the NumPy array's elements are floating-point numbers.

Callout: In the Corner

What may also surprise you is that when Python displays an array, it shows the element with index `[0, 0]` in the upper left corner rather than the lower left. This is consistent with the way mathematicians draw matrices but different from the Cartesian coordinates. The indices are (row, column) instead of (column, row) for the same reason, which can be confusing when plotting data.

Callout: Not All Functions Have Input

Generally, a function uses inputs to produce outputs. However, some functions produce outputs without needing any input. For example, checking the current time doesn't require any input.

Python:

```
import time
```

```
print(time.ctime())
```

Output:

```
Sat Mar 26 13:07:33 2016
```

For functions that don't take in any arguments, we still need parentheses (`()`) to tell Python to go and do something for us.

Callout: Some IPython Magic

If you're using a Jupyter notebook, you'll need to execute the following command in order for your matplotlib images to appear in the notebook when `show()` is called:

Python:

```
%matplotlib inline
```

The `%` indicates an IPython magic function - a function that is only valid within the notebook environment. Note that you only have to execute this function once per notebook.

Grouping plots

Exercise: Check Your Understanding

What values do the variables `mass` and `age` have after each statement in the following program? Test your answers by executing the commands.

Python:

```
mass = 47.5
```

```
age = 122
mass = mass * 2.0
age = age - 20
print(mass, age)
```

Exercise: Sorting Out References

What does the following program print out?

Python:

```
first, second = 'Grace', 'Hopper'
third, fourth = second, first
print(third, fourth)
```

Exercise: Slicing Strings

A section of an array is called a slice. We can take slices of character strings as well:

Python:

```
element = 'oxygen'

print('first three characters:', element[0:3])

print('last three characters:', element[3:6])
```

Output:

```
first three characters: oxy

last three characters: gen
```

What is the value of `element[:4]`? What about `element[4:]`? Or `element[:]`?

What is `element[-1]`? What is `element[-2]`?

Given those answers, explain what `element[1:-1]` does.

Exercise: Thin Slices

The expression `element[3:3]` produces an empty string, i.e., a string that contains no characters. If `data` holds our array of patient data, what does `data[3:3, 4:4]` produce? What about `data[3:3, :]`?

Exercise: Plot Scaling

Why do all of our plots stop just short of the upper end of our graph?

If we want to change this, we can use the `set_ylim(min, max)` method of each 'axes', for example:

Python:

```
axes3.set_ylim(0,6)
```

Update your plotting code to automatically set a more appropriate scale. (Hint: you can make use of the `max` and `min` methods to help.)

Exercise: Drawing Straight Lines

In the center and right subplots above, we expect all lines to look like step functions because non-integer value are not realistic for the minimum and maximum values. However, you can see that the lines are not always vertical or horizontal, and in particular the step function in the subplot on the right looks slanted. Why is this?

Exercise: Make Your Own Plot

Create a plot showing the standard deviation (`numpy.std`) of the inflammation data for each day across all patients.

Exercise: Moving Plots Around

Modify the program to display the three plots on top of one another instead of side by side.

Exercise: Stacking Arrays

Arrays can be concatenated and stacked on top of one another, using NumPy's `vstack` and `hstack` functions for vertical and horizontal stacking, respectively.

Python:

```
import numpy


A = numpy.array([[1,2,3], [4,5,6], [7, 8, 9]])
print('A = ')
print(A)


B = numpy.hstack([A, A])
print('B = ')
print(B)


C = numpy.vstack([A, A])
print('C = ')
print(C)
```

Output:

```
A =
[[1 2 3]
 [4 5 6]
 [7 8 9]]
B =
[[1 2 3 1 2 3]
 [4 5 6 4 5 6]
 [7 8 9 7 8 9]]
C =
[[1 2 3]
 [4 5 6]
 [7 8 9]
 [1 2 3]
 [4 5 6]
 [7 8 9]]
```

Write some additional code that slices the first and last columns of `A`, and stacks them into a 3x2 array. Make sure to `print` the results to verify your solution.

Exercise: Change In Inflammation

This patient data is *longitudinal* in the sense that each row represents a series of observations relating to one individual. This means that the change in inflammation over time is a meaningful concept.

The `numpy.diff()` function takes a NumPy array and returns the differences between two successive values along a specified axis. For example, a NumPy array that looks like this:

Python:

```
npdiff = numpy.array([ 0,  2,  5,  9, 14])
```

Calling `numpy.diff(npdiff)` would do the following calculations and put the answers in another array.

Python:

```
[ 2 - 0, 5 - 2, 9 - 5, 14 - 9 ]
```

Python:

```
numpy.diff(npdiff)
```

Python:

```
array([2, 3, 4, 5])
```

Which axis would it make sense to use this function along?

If the shape of an individual data file is `(60, 40)` (60 rows and 40 columns), what would the shape of the array be after you run the `diff()` function and why?

How would you find the largest change in inflammation for each patient? Does it matter if the change in inflammation is an increase or a decrease?

Keypoints:

- Import a library into a program using `import libraryname`.
- Use the `numpy` library to work with arrays in Python.
- Use `variable = value` to assign a value to a variable in order to record it in memory.
- Variables are created on demand whenever a value is assigned to them.
- Use `print(something)` to display the value of `something`.
- The expression `array.shape` gives the shape of an array.
- Use `array[x, y]` to select a single element from a 2D array.
- Array indices start at 0, not 1.
- Use `low:high` to specify a `slice` that includes the indices from `low` to `high-1`.
- All the indexing and slicing that works on arrays also works on strings.
- Use `# some kind of explanation` to add comments to programs.
- Use `numpy.mean(array)`, `numpy.max(array)`, and `numpy.min(array)` to calculate simple statistics.
- Use `numpy.mean(array, axis=0)` or `numpy.mean(array, axis=1)` to calculate statistics across the specified axis.
- Use the `pyplot` library from `matplotlib` for creating simple visualizations.

Repeating Actions with Loops

https://denubis.github.io/python-novice-inflammation/02-loop/index.html

Questions:

- How can I do the same operations on many different values?

Objectives:

- Explain what a `for` loop does.
- Correctly write `for` loops to repeat simple calculations.
- Trace changes to a loop variable as the loop runs.
- Trace changes to other variables as they are updated by a `for` loop.

Callout: What's in a name?

In the example above, the loop variable was given the name `char` as a mnemonic; it is short for 'character'. We can choose any name we want for variables. We might just as easily have chosen the name `banana` for the loop variable, as long as we use the same name when we invoke the variable inside the loop:

Python:

```
word = 'oxygen'
```

```
for banana in word:
    print(banana)
```

Output:

```
o
x
y
g
e
n
```

It is a good idea to choose variable names that are meaningful, otherwise it would be more difficult to understand what the loop is doing.

Exercise: From 1 to N

Python has a built-in function called `range` that generates a sequence of numbers. `range` can accept 1, 2, or 3 parameters.

- If one parameter is given, `range` generates a sequence of that length, starting at zero and incrementing by 1. For example, `range(3)` produces the numbers `0, 1, 2`.
- If two parameters are given, `range` starts at the first and ends just before the second, incrementing by one. For example, `range(2, 5)` produces `2, 3, 4`.

- If `range` is given 3 parameters, it starts at the first one, ends just before the second one, and increments by the third one. For example, `range(3, 10, 2)` produces `3, 5, 7, 9`.

Using `range`, write a loop that uses `range` to print the first 3 natural numbers:

Python:

```
1

2

3
```

Exercise: Understanding the loops

Given the following loop:

```
word = 'oxygen'

for char in word:

    print(char)
```

How many times is the body of the loop executed?

- 3 times
- 4 times
- 5 times

- 6 times

Exercise: Computing Powers With Loops

Exponentiation is built into Python:

Python:

```
print(5 ** 3)
```

Output:

```
125
```

Write a loop that calculates the same result as `5 ** 3` using multiplication (and without exponentiation).

Exercise: Reverse a String

Knowing that two strings can be concatenated using the `+` operator, write a loop that takes a string and produces a new string with the characters in reverse order, so `'Newton'` becomes `'notweN'`.

```
# Expected output: 'Newton' becomes 'notweN'.


input_string = 'Newton'

output_string = ''

print(input_string[1], input_string[0])

for letter in input_string:

        # My error was putting output_string *first* and then adding letter.

        # I should instead add letter to the front of output

        # string (or rather, add output string to letter)

        output_string = letter + output_string

print(output_string)
```

Exercise: Computing the Value of a Polynomial

The built-in function `enumerate` takes a sequence (e.g. a list) and generates a new sequence of the same length. Each element of the new sequence is a pair composed of the index (0, 1, 2,…) and the value from the original sequence:

Python:

```
for i, x in enumerate(xs):
    # Do something using i and x
```

The code above loops through `xs`, assigning the index to `i` and the value to `x`.

Suppose you have encoded a polynomial as a list of coefficients in the following way: the first element is the constant term, the second element is the coefficient of the linear term, the third is the coefficient of the quadratic term, etc.

Python:

```
x = 5
```

```
cc = [2, 4, 3]
```

Output:

```
y = cc[0] * x**0 + cc[1] * x**1 + cc[2] * x**2
```

```
y = 97
```

Write a loop using `enumerate(cc)` which computes the value `y` of any polynomial, given `x` and `cc`.

Keypoints:

- Use `for variable in sequence` to process the elements of a sequence one at a time.
- The body of a `for` loop must be indented.
- Use `len(thing)` to determine the length of something that contains other values.

Storing Multiple Values in Lists

https://denubis.github.io/python-novice-inflammation/03-lists/index.html

Questions:

- How can I store many values together?

Objectives:

- Explain what a list is.
- Create and index lists of simple values.
- Change the values of individual elements
- Append values to an existing list
- Reorder and slice list elements
- Create and manipulate nested lists

Callout: Ch-Ch-Ch-Ch-Changes

Data which can be modified in place is called mutable, while data which cannot be modified is called immutable. Strings and numbers are immutable. This does not mean that variables with string or number values are constants, but when we want to change the value of a string or number variable, we can only replace the old value with a completely new value.

Lists and arrays, on the other hand, are mutable: we can modify them after they have been created. We can change individual elements, append new elements, or reorder the whole list. For some operations, like sorting, we can choose whether to use a function that modifies the data in-place or a function that returns a modified copy and leaves the original unchanged.

Be careful when modifying data in-place. If two variables refer to the same list, and you modify the list value, it will change for both variables!

Python:

```
salsa = ['peppers', 'onions', 'cilantro', 'tomatoes']

my_salsa = salsa        # <-- my_salsa and salsa point to the *same* list data in memory

salsa[0] = 'hot peppers'

print('Ingredients in my salsa:', my_salsa)
```

Output:

```
Ingredients in my salsa: ['hot peppers', 'onions', 'cilantro', 'tomatoes']
```

If you want variables with mutable values to be independent, you must make a copy of the value when you assign it.

Python:

```
salsa = ['peppers', 'onions', 'cilantro', 'tomatoes']

my_salsa = list(salsa)        # <-- makes a *copy* of the list

salsa[0] = 'hot peppers'

print('Ingredients in my salsa:', my_salsa)
```

Output:

```
Ingredients in my salsa: ['peppers', 'onions', 'cilantro', 'tomatoes']
```

Because of pitfalls like this, code which modifies data in place can be more difficult to understand. However, it is often far more efficient to modify a large data structure in place than to create a modified copy for every small change. You should consider both of these aspects when writing your code.

Callout: Nested Lists

Since lists can contain any Python variable, it can even contain other lists.

For example, we could represent the products in the shelves of a small grocery shop:

Python:

```
x = [['pepper', 'zucchini', 'onion'],
     ['cabbage', 'lettuce', 'garlic'],
     ['apple', 'pear', 'banana']]
```

Here is a visual example of how indexing a list of lists `x` works:

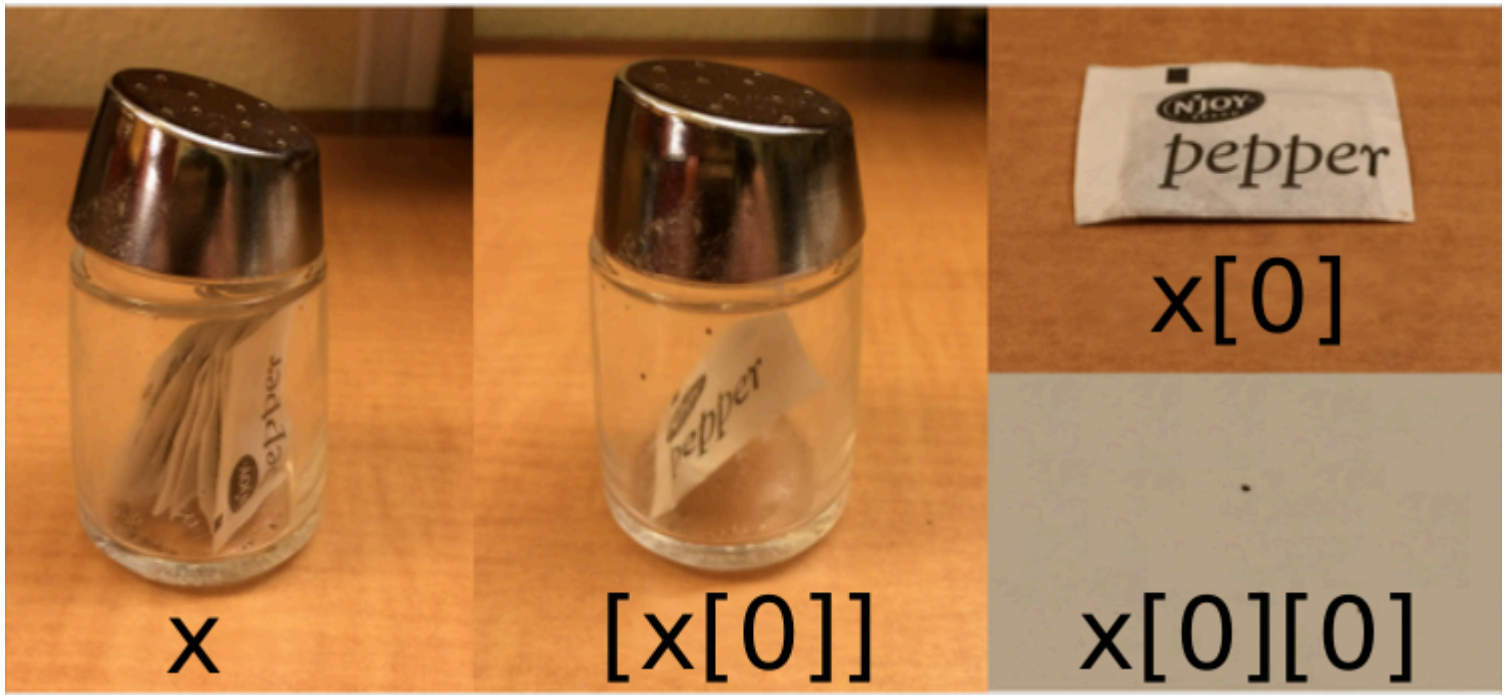

Image: The first element of a list. Adapted from @hadleywickham. https://denubis.github.io/python-novice-inflammation/fig/indexing_lists_python.png

Using the previously declared list `x`, these would be the results of the index operations shown in the image:

Python:

```
print([x[0]])
```

Output:

```
[['pepper', 'zucchini', 'onion']]
```

Python:

```
print(x[0])
```

Output:

```
['pepper', 'zucchini', 'onion']
```

Python:

```
print(x[0][0])
```

Output:

```
'pepper'
```

Thanks to Hadley Wickham for the image above.

Callout: Heterogeneous Lists

Lists in Python can contain elements of different types. Example:

Python:

```
sample_ages = [10, 12.5, 'Unknown']
```

Exercise: Turn a String Into a List

Use a for-loop to convert the string “hello” into a list of letters:

Python:

```
["h", "e", "l", "l", "o"]
```

Hint: You can create an empty list like this:

Python:

```
my_list = []
```

Exercise: Slicing From the End

Use slicing to access only the last four characters of a string or entries of a list.

Python:

```
string_for_slicing = "Observation date: 02-Feb-2013"

list_for_slicing = [["fluorine", "F"],
                    ["chlorine", "Cl"],
                    ["bromine", "Br"],
                    ["iodine", "I"],
                    ["astatine", "At"]]
```

Output:

```
"2013"
```

```
[["chlorine", "Cl"], ["bromine", "Br"], ["iodine", "I"], ["astatine", "At"]]
```

Would your solution work regardless of whether you knew beforehand the length of the string or list (e.g. if you wanted to apply the solution to a set of lists of different lengths)? If not, try to change your approach to make it more robust.

Hint: Remember that indices can be negative as well as positive

Exercise: Non-Continuous Slices

So far we've seen how to use slicing to take single blocks of successive entries from a sequence. But what if we want to take a subset of entries that aren't next to each other in the sequence?

You can achieve this by providing a third argument to the range within the brackets, called the *step size*. The example below shows how you can take every third entry in a list:

Python:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

```
subset = primes[0:12:3]
```

```
print("subset", subset)
```

Output:

```
subset [2, 7, 17, 29]
```

Notice that the slice taken begins with the first entry in the range, followed by entries taken at equally-spaced intervals (the steps) thereafter. If you wanted to begin the subset with the third entry, you would need to specify that as the starting point of the sliced range:

Python:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]

subset = primes[2:12:3]

print("subset", subset)
```

Output:

```
subset [5, 13, 23, 37]
```

Use the step size argument to create a new string that contains only every other character in the string “In an octopus’s garden in the shade”

Python:

```
beatles = "In an octopus's garden in the shade"
```

Output:

```
I notpssgre ntesae
```

Exercise: Overloading

`+` usually means addition, but when used on strings or lists, it means “concatenate”. Given that, what do you think the multiplication operator `*` does on lists? In particular, what will be the output of the following code?

Python:

```
counts = [2, 4, 6, 8, 10]

repeats = counts * 2

print(repeats)
```

- `[2, 4, 6, 8, 10, 2, 4, 6, 8, 10]`
- `[4, 8, 12, 16, 20]`
- `[[2, 4, 6, 8, 10],[2, 4, 6, 8, 10]]`
- `[2, 4, 6, 8, 10, 4, 8, 12, 16, 20]`

The technical term for this is *operator overloading*: a single operator, like `+` or `*`, can do different things depending on what it's applied to.

Keypoints:

- `[value1, value2, value3, ...]` creates a list.
- Lists can contain any Python object, including lists (i.e., list of lists).
- Lists are indexed and sliced with square brackets (e.g., list[0] and list[2:9]), in the same way as strings and arrays.
- Lists are mutable (i.e., their values can be changed in place).
- Strings are immutable (i.e., the characters in them cannot be changed).

Analyzing Data from Multiple Files

https://denubis.github.io/python-novice-inflammation/04-files/index.html

Questions:

- How can I do the same operations on many different files?

Objectives:

- Use a library function to get a list of filenames that match a wildcard pattern.

- Write a `for` loop to process multiple files.

Topics covered in Exercise 2

Processing multiple files

- Glob.glob (File name using the * character as a ” wild card”)
- Sorted(glob.glob(filename*)): sort files in order
- Make sure that the glob.glob generate correct filenames that are in the folder
- Do all the code to generate figures in one cell
- To delete empty cell, click on the cell, when the bar is blue, press “d” twice

Exercise: Plotting Differences

Plot the difference between the average of the first dataset and the average of the second dataset, i.e., the difference between the leftmost plot of the first two figures.

Exercise: Generate Composite Statistics

Use each of the files once to generate a dataset containing values averaged over all patients:

Python:

```
filenames = glob.glob('inflammation*.csv')

composite_data = numpy.zeros((60,40))

for f in filenames:

    # sum each new file's data into composite_data as it's read

    #

# and then divide the composite_data by number of samples
```

```
composite_data /= len(filenames)
```

Then use pyplot to generate average, max, and min for all patients.

Keypoints:

- Use `glob.glob(pattern)` to create a list of files whose names match a pattern.
- Use `*` in a pattern to match zero or more characters, and `?` to match any single character.
- Adding title to plots and label:
- for file in filenames:
- data = numpy.loadtxt(fname = file, delimiter = ',')
- fig = matplotlib.pyplot.figure(figsize=(10.0,3.0))
- fig.suptitle (file, y = 1.02)
- subplot1 = fig.add_subplot(1,3,1) # Note: Number refer to position, number of rows, number in column
- subplot2 = fig.add_subplot(1,3,2)
- subplot3 = fig.add_subplot(1,3,3)
- subplot1.set_ylabel('Patients')
- subplot1.set_xlabel('Day')
- subplot1.set_title('Daily average')
- subplot1.plot(numpy.mean(data, axis=0))
- subplot2.set_ylabel('Daily average')
- subplot2.plot(numpy.max(data, axis=0))
- subplot3.set_ylabel('Daily average')
- subplot3.plot(numpy.min(data, axis=0))
- fig.tight_layout()
- matplotlib.pyplot.show()
- Also you can use “fig.suptitle” to make a title for the whole figure (not the subplot).
-

Making Choices

https://denubis.github.io/python-novice-inflammation/05-cond/index.html

Questions:

- How can my programs do different things based on data values?

Objectives:

- Write conditional statements including `if`, `elif`, and `else` branches.
- Correctly evaluate expressions containing `and` and `or`.

Topics covered in Exercise 2

Conditionals

- If - else
- if - elif
- If and (tests if both are true)
- If or (tests if at least one is true)l
- Use an == sign to test for equality.

Callout: True and False

`True` and `False` are special words in Python called `booleans`, which represent truth values. A statement such as `1 < 0` returns the value `False`, while `-1 < 0` returns the value `True`.

Checking our Data

Exercise: How Many Paths?

Consider this code:

Python:

```
if 4 > 5:
    print('A')
elif 4 == 5:
    print('B')
elif 4 < 5:
    print('C')
```

Which of the following would be printed if you were to run this code? Why did you pick this answer?

- A
- B

- C
- B and C

Exercise: What Is Truth?

`True` and `False` booleans are not the only values in Python that are true and false. In fact, *any* value can be used in an `if` or `elif`. After reading and running the code below, explain what the rule is for which values are considered true and which are considered false.

Python:

```
if '':
    print('empty string is true')
if 'word':
    print('word is true')
if []:
    print('empty list is true')
if [1, 2, 3]:
    print('non-empty list is true')
if 0:
    print('zero is true')
if 1:
    print('one is true')
```

Exercise: That's Not Not What I Meant

Sometimes it is useful to check whether some condition is not true. The Boolean operator `not` can do this explicitly. After reading and running the code below, write some `if` statements that use `not` to test the rule that you formulated in the previous challenge.

Python:

```
if not '':
    print('empty string is not true')
if not 'word':
    print('word is not true')
if not not True:
    print('not not True is true')
```

Exercise: Close Enough

Write some conditions that print `True` if the variable `a` is within 10% of the variable `b` and `False` otherwise. Compare your implementation with your partner's: do you get the same answer for all possible pairs of numbers?

Exercise: In-Place Operators

Python (and most other languages in the C family) provides in-place operators that work like this:

Python:

```
x = 1  # original value

x += 1 # add one to x, assigning result back to x

x *= 3 # multiply x by 3

print(x)
```

Output:

```
6
```

Write some code that sums the positive and negative numbers in a list separately, using in-place operators. Do you think the result is more or less readable than writing the same without in-place operators?

Exercise: Sorting a List Into Buckets

In our `data` folder, large data sets are stored in files whose names start with “inflammation-“ and small data sets – in files whose names start with “small-“. We also have some other files that we do not care about at this point. We’d like to break all these files into three lists called `large_files`, `small_files`, and `other_files`, respectively.

Add code to the template below to do this. Note that the string method `startswith` returns `True` if and only if the string it is called on starts with the string passed as an argument, that is:

Python:

```
"String".startswith("Str")
```

Output:

```
True
```

But

Python:

```
"String".startswith("str")
```

Output:

```
False
```

Use the following Python code as your starting point:

Python:

```
files = ['inflammation-01.csv',
         'myscript.py',
         'inflammation-02.csv',
         'small-01.csv',
         'small-02.csv']
large_files = []
small_files = []
other_files = []
```

Your solution should:

- loop over the names of the files
- figure out which group each filename belongs
- append the filename to that list

In the end the three lists should be:

Python:

```
large_files = ['inflammation-01.csv', 'inflammation-02.csv']
small_files = ['small-01.csv', 'small-02.csv']
```

```
other_files = ['myscript.py']
```

Exercise: Counting Vowels

- Write a loop that counts the number of vowels in a character string.
- Test it on a few individual words and full sentences.
- Once you are done, compare your solution to your neighbor's. Did you make the same decisions about how to handle the letter 'y' (which some people think is a vowel, and some do not)?

Keypoints:

- Use `if condition` to start a conditional statement, `elif condition` to provide additional tests, and `else` to provide a default.
- The bodies of the branches of conditional statements must be indented.
- Use `==` to test for equality.
- `X and Y` is only true if both `X` and `Y` are true.
- `X or Y` is true if either `X` or `Y`, or both, are true.
- Zero, the empty string, and the empty list are considered false; all other numbers, strings, and lists are considered true.
- `True` and `False` represent truth values.

Creating Functions

https://denubis.github.io/python-novice-inflammation/06-func/index.html

Questions:

- How can I define new functions?
- What's the difference between defining and calling a function?
- What happens when I call a function?

Objectives:

- Define a function that takes parameters.
- Return a value from a function.
- Test and debug a function.
- Set default values for function parameters.
- Explain why we should divide programs into small, single-purpose functions.

Composing Functions

Def func_name(variable)

    Return value

Tidying up

Testing and Documenting

Defining Defaults

Readable functions

Exercise: Combining Strings

“Adding” two strings produces their concatenation: `'a' + 'b'` is `'ab'`. Write a function called `fence` that takes two parameters called `original` and `wrapper` and returns a new string that has the wrapper character at the beginning and end of the original. A call to your function should look like this:

Python:

```
print(fence('name', '*'))
```

Output:

```
*name*
```

Exercise: Return versus print

Note that `return` and `print` are not interchangeable. `print` is a Python function that *prints* data to the screen. It enables us, *users*, see the data. `return` statement, on the other hand, makes data visible to the program. Let's have a look at the following function:

Python:

```
def add(a, b):
    print(a + b)
```

Question: What will we see if we execute the following commands?

Python:

```
A = add(7, 3)
print(A)
```

Exercise: Selecting Characters From Strings

If the variable `s` refers to a string, then `s[0]` is the string's first character and `s[-1]` is its last. Write a function called `outer` that returns a string made up of just the first and last characters of its input. A call to your function should look like this:

Python:

```
print(outer('helium'))
```

Output:

```
hm
```

Exercise: Rescaling an Array

Write a function `rescale` that takes an array as input and returns a corresponding array of values scaled to lie in the range 0.0 to 1.0. (Hint: If `L` and `H` are the lowest and highest values in the original array, then the replacement for a value `v` should be `(v-L) / (H-L)`.)

Exercise: Testing and Documenting Your Function

Run the commands `help(numpy.arange)` and `help(numpy.linspace)` to see how to use these functions to generate regularly-spaced values, then use those values to test your `rescale` function. Once you've successfully tested your function, add a docstring that explains what it does.

Exercise: Defining Defaults

Rewrite the `rescale` function so that it scales data to lie between `0.0` and `1.0` by default, but will allow the caller to specify lower and upper bounds if they want. Compare your implementation to your neighbor's: do the two functions always behave the same way?

Exercise: Variables Inside and Outside Functions

What does the following piece of code display when run — and why?

Python:

```
f = 0

k = 0


def f2k(f):

    k = ((f-32)*(5.0/9.0)) + 273.15

    return k


f2k(8)

f2k(41)

f2k(32)


print(k)
```

Exercise: Mixing Default and Non-Default Parameters

Given the following code:

Python:

```
def numbers(one, two=2, three, four=4):
    n = str(one) + str(two) + str(three) + str(four)
    return n


print(numbers(1, three=3))
```

what do you expect will be printed? What is actually printed? What rule do you think Python is following?

- `1234`
- `one2three4`
- `1239`
- `SyntaxError`

Given that, what does the following piece of code display when run?

Python:

```
def func(a, b=3, c=6):
```

```
    print('a: ', a, 'b: ', b, 'c:', c)

func(-1, 2)
```

- `a: b: 3 c: 6`
- `a: -1 b: 3 c: 6`
- `a: -1 b: 2 c: 6`
- `a: b: -1 c: 2`

Exercise: The Old Switcheroo

Consider this code:

Python:

```
a = 3

b = 7


def swap(a, b):

    temp = a

    a = b

    b = temp


swap(a, b)
```

```
print(a, b)
```

Which of the following would be printed if you were to run this code? Why did you pick this answer?

- `7 3`
- `3 7`
- `3 3`
- `7 7`

Exercise: Readable Code

Revise a function you wrote for one of the previous exercises to try to make the code more readable. Then, collaborate with one of your neighbors to critique each other's functions and discuss how your function implementations could be further improved to make them more readable.

Keypoints:

- Define a function using `def function_name(parameter)`.
- The body of a function must be indented.
- Call a function using `function_name(value)`.
- Numbers are stored as integers or floating-point numbers.
- Variables defined within a function can only be seen and used within the body of the function.
- If a variable is not defined within the function it is used, Python looks for a definition before the function call
- Use `help(thing)` to view help for something.
- Put docstrings in functions to provide help for that function.
- Specify default values for parameters when defining a function using `name=value` in the parameter list.

- Parameters can be passed by matching based on name, by position, or by omitting them (in which case the default value is used).
- Put code whose parameters change frequently in a function, then call it with different parameter values to customize its behavior.

Errors and Exceptions

https://denubis.github.io/python-novice-inflammation/07-errors/index.html

Questions:

- How does Python report errors?
- How can I handle errors in Python programs?

Objectives:

- To be able to read a traceback, and determine where the error took place and what type it is.
- To be able to describe the types of situations in which syntax errors, indentation errors, name errors, index errors, and missing file errors occur.

Callout: Long Tracebacks

Sometimes, you might see a traceback that is very long – sometimes they might even be 20 levels deep! This can make it seem like something horrible happened, but really it just means that your program called many functions before it ran into the error. Most of the time, you can just pay attention to the bottom-most level, which is the actual place where the error occurred.

Syntax Errors

Callout: Tabs and Spaces

Some indentation errors are harder to spot than others. In particular, mixing spaces and tabs can be difficult to spot because they are both whitespace. In the example below, the first two lines in the body of the function `some_function` are indented with tabs, while the third line — with spaces. If you're working in a Jupyter notebook, be sure to copy and paste this example rather than trying to type it in manually because Jupyter automatically replaces tabs with spaces.

Python:

```
def some_function():
	msg = "hello, world!"
	print(msg)
	  return msg
```

Visually it is impossible to spot the error. Fortunately, Python does not allow you to mix tabs and spaces.

```
 File "<ipython-input-5-653b36fbcd41>", line 4
    return msg
               ^
TabError: inconsistent use of tabs and spaces in indentation
```

Variable Name Errors

Index Errors

File Errors

Exercise: Reading Error Messages

Read the Python code and the resulting traceback below, and answer the following questions:

- How many levels does the traceback have?
- What is the function name where the error occurred?
- On which line number in this function did the error occur?
- What is the type of error?
- What is the error message?

Python:

```
# This code has an intentional error. Do not type it directly;
# use it for reference to understand the error message below.
def print_message(day):
    messages = {
        "monday": "Hello, world!",
```

```
        "tuesday": "Today is Tuesday!",
        "wednesday": "It is the middle of the week.",
        "thursday": "Today is Donnerstag in German!",
        "friday": "Last day of the week!",
        "saturday": "Hooray for the weekend!",
        "sunday": "Aw, the weekend is almost over."
    }
    print(messages[day])

def print_friday_message():
    print_message("Friday")

print_friday_message()
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-1-4be1945adbe2> in <module>()
     14     print_message("Friday")
     15
---> 16 print_friday_message()

<ipython-input-1-4be1945adbe2> in print_friday_message()
     12
```

```
     13 def print_friday_message():
---> 14     print_message("Friday")
     15
     16 print_friday_message()


<ipython-input-1-4be1945adbe2> in print_message(day)
      9         "sunday": "Aw, the weekend is almost over."
     10     }
---> 11     print(messages[day])
     12
     13 def print_friday_message():


KeyError: 'Friday'
```

Exercise: Identifying Syntax Errors

- Read the code below, and (without running it) try to identify what the errors are.
- Run the code, and read the error message. Is it a `SyntaxError` or an `IndentationError`?
- Fix the error.
- Repeat steps 2 and 3, until you have fixed all the errors.

Python:

```
def another_function
  print("Syntax errors are annoying.")
   print("But at least Python tells us about them!")
  print("So they are usually not too hard to fix.")
```

Exercise: Identifying Variable Name Errors

- Read the code below, and (without running it) try to identify what the errors are.
- Run the code, and read the error message. What type of `NameError` do you think this is? In other words, is it a string with no quotes, a misspelled variable, or a variable that should have been defined but was not?
- Fix the error.
- Repeat steps 2 and 3, until you have fixed all the errors.

Python:

```
for number in range(10):
    # use a if the number is a multiple of 3, otherwise use b
    if (Number % 3) == 0:
        message = message + a
    else:
        message = message + "b"
print(message)
```

Exercise: Identifying Index Errors

- Read the code below, and (without running it) try to identify what the errors are.
- Run the code, and read the error message. What type of error is it?
- Fix the error.bin

Python:

```
seasons = ['Spring', 'Summer', 'Fall', 'Winter']

print('My favorite season is ', seasons[4])
```

Keypoints:

- Tracebacks can look intimidating, but they give us a lot of useful information about what went wrong in our program, including where the error occurred and what type of error it was.
- An error having to do with the 'grammar' or syntax of the program is called a `SyntaxError`. If the issue has to do with how the code is indented, then it will be called an `IndentationError`.
- A `NameError` will occur if you use a variable that has not been defined, either because you meant to use quotes around a string, you forgot to define the variable, or you just made a typo.
- Containers like lists and strings will generate errors if you try to access items in them that do not exist. This type of error is called an `IndexError`.

- Trying to read a file that does not exist will give you an `FileNotFoundError`. Trying to read a file that is open for writing, or writing to a file that is open for reading, will give you an `IOError`.

Defensive Programming

https://denubis.github.io/python-novice-inflammation/08-defensive/index.html

Questions:

- How can I make my programs more reliable?

Objectives:

- Explain what an assertion is.
- Add assertions that check the program's state is correct.
- Correctly add precondition and postcondition assertions to functions.
- Explain what test-driven development is, and use it when creating new functions.
- Explain why variables should be initialized using actual data values rather than arbitrary constants.

Assertions

Test-Driven Development

Exercise: Pre- and Post-Conditions

Suppose you are writing a function called `average` that calculates the average of the numbers in a list. What pre-conditions and post-conditions would you write for it? Compare your answer to your neighbor's: can you think of a function that will pass your tests but not his/hers or vice versa?

Exercise: Testing Assertions

Given a sequence of a number of cars, the function `get_total_cars` returns the total number of cars.

Python:

```
get_total_cars([1, 2, 3, 4])
```

Output:

```
10
```

Python:

```
get_total_cars(['a', 'b', 'c'])
```

Output:

```
ValueError: invalid literal for int() with base 10: 'a'
```

Explain in words what the assertions in this function check, and for each one, give an example of input that will make that assertion fail.

Python:

```
def get_total(values):
    assert len(values) > 0
    for element in values:
        assert int(element)
    values = [int(element) for element in values]
    total = sum(values)
    assert total > 0
    return total
```

Exercise: Fixing and Testing

Fix `range_overlap`. Re-run `test_range_overlap` after each change you make.

Keypoints:

- Program defensively, i.e., assume that errors are going to arise, and write code to detect them when they do.
- Put assertions in programs to check their state as they run, and to help readers understand how those programs are supposed to work.
- Use preconditions to check that the inputs to a function are safe to use.
- Use postconditions to check that the output from a function is safe to use.
- Write tests before writing code in order to help determine exactly what that code is supposed to do.

Debugging

https://denubis.github.io/python-novice-inflammation/09-debugging/index.html

Questions:

- How can I debug my program?

Objectives:

- Debug code containing an error systematically.
- Identify ways of making code less error-prone and more easily tested.

Know What It's Supposed to Do

Make It Fail Every Time

Make It Fail Fast

Change One Thing at a Time, For a Reason

Keep Track of What You've Done

Callout: Version Control Revisited

Version control is often used to reset software to a known state during debugging, and to explore recent changes to code that might be responsible for bugs. In particular, most version control systems (e.g. git, Mercurial) have:

- a `blame` command that shows who last changed each line of a file;
- a `bisect` command that helps with finding the commit that introduced an issue.

Be Humble

Exercise: Debug With a Neighbor

Take a function that you have written today, and introduce a tricky bug. Your function should still run, but will give the wrong output. Switch seats with your neighbor and attempt to debug the bug that they introduced into their function. Which of the principles discussed above did you find helpful?

Exercise: Not Supposed to be the Same

You are assisting a researcher with Python code that computes the Body Mass Index (BMI) of patients. The researcher is concerned because all patients seemingly have unusual and identical BMIs, despite having different physiques. BMI is calculated as **weight in kilograms** divided by the square of **height in metres**.

Use the debugging principles in this exercise and locate problems with the code. What suggestions would you give the researcher for ensuring any later changes they make work correctly?

Python:

```
patients = [[70, 1.8], [80, 1.9], [150, 1.7]]


def calculate_bmi(weight, height):
    return weight / (height ** 2)


for patient in patients:
    weight, height = patients[0]
    bmi = calculate_bmi(height, weight)
    print("Patient's BMI is: %f" % bmi)
```

Output:

```
Patient's BMI is: 0.000367
```

```
Patient's BMI is: 0.000367
```

```
Patient's BMI is: 0.000367
```

Keypoints:

- Know what code is supposed to do *before* trying to debug it.
- Make it fail every time.
- Make it fail fast.
- Change one thing at a time, and for a reason.
- Keep track of what you've done.
- Be humble.

Command-Line Programs

https://denubis.github.io/python-novice-inflammation/10-cmdline/index.html

Questions:

- How can I write Python programs that will work like Unix command-line tools?

Objectives:

- Use the values of command-line arguments in a program.
- Handle flags and files separately in a command-line program.
- Read data from standard input in a program so that it can be used in a pipeline.

Callout: Switching to Shell Commands

In this lesson we are switching from typing commands in a Python interpreter to typing commands in a shell terminal window (such as bash). When you see a `$` in front of a command that tells you to run that command in the shell rather than the Python interpreter.

Command-Line Arguments

Callout: Running Versus Importing

Running a Python script in bash is very similar to importing that file in Python. The biggest difference is that we don't expect anything to happen when we import a file, whereas when running a script, we expect to see some output printed to the console.

In order for a Python script to work as expected when imported or when run as a script, we typically put the part of the script that produces output in the following if statement:

Python:

```
if __name__ == '__main__':

    main()  # Or whatever function produces output
```

When you import a Python file, `__name__` is the name of that file (e.g., when importing `readings.py`, `__name__` is `'readings'`). However, when running a script in bash, `__name__` is always set to `'__main__'` in that script so that you can determine if the file is being imported or run as a script.

Callout: The Right Way to Do It

If our programs can take complex parameters or multiple filenames, we shouldn't handle `sys.argv` directly. Instead, we should use Python's `argparse` library, which handles common cases in a systematic way, and also makes it easy for us to provide sensible error messages for our users. We will not cover this module in this lesson but you can go to Tshepang Lekhonkhobe's Argparse tutorial that is part of Python's Official Documentation.

Handling Multiple Files

Callout: The Right Way to Do It

At this point, we have created three versions of our script called `readings_01.py`, `readings_02.py`, and `readings_03.py`. We wouldn't do this in real life: instead, we would have one file called `readings.py` that we committed to version control every time we got an enhancement working. For teaching, though, we need all the successive versions side by side.

Handling Command-Line Flags

Handling Standard Input

Exercise: Arithmetic on the Command Line

Write a command-line program that does addition and subtraction:

Code:

```
$ python arith.py add 1 2
```

Output:

```
3
```

Code:

```
$ python arith.py subtract 3 4
```

Output:

```
-1
```

Exercise: Finding Particular Files

Using the `glob` module introduced earlier, write a simple version of `ls` that shows files in the current directory with a particular suffix. A call to this script should look like this:

Code:

```
$ python my_ls.py py
```

Output:

```
left.py
```

```
right.py
```

```
zero.py
```

Exercise: Changing Flags

Rewrite `readings.py` so that it uses `-n`, `-m`, and `-x` instead of `--min`, `--mean`, and `--max` respectively. Is the code easier to read? Is the program easier to understand?

Exercise: Adding a Help Message

Separately, modify `readings.py` so that if no parameters are given (i.e., no action is specified and no filenames are given), it prints a message explaining how it should be used.

Exercise: Adding a Default Action

Separately, modify `readings.py` so that if no action is given it displays the means of the data.

Exercise: A File-Checker

Write a program called `check.py` that takes the names of one or more inflammation data files as arguments and checks that all the files have the same number of rows and columns. What is the best way to test your program?

Exercise: Counting Lines

Write a program called `line_count.py` that works like the Unix `wc` command:

- If no filenames are given, it reports the number of lines in standard input.
- If one or more filenames are given, it reports the number of lines in each, followed by the total number of lines.

Exercise: Generate an Error Message

Write a program called `check_arguments.py` that prints usage then exits the program if no arguments are provided. (Hint: You can use `sys.exit()` to exit the program.)

Code:

```
$ python check_arguments.py
```

Output:

```
usage: python check_argument.py filename.txt
```

Code:

```
$ python check_arguments.py filename.txt
```

Output:

```
Thanks for specifying arguments!
```

Keypoints:

- The `sys` library connects a Python program to the system it is running on.
- The list `sys.argv` contains the command-line arguments that a program was run with.
- Avoid silent failures.
- The pseudo-file `sys.stdin` connects to a program's standard input.
- The pseudo-file `sys.stdout` connects to a program's standard output.

BEFORE YOU LEAVE

Please fill out the post-training survey