Data Structures

BILD 62

From last week

- There are multiple ways to write and run Python code
- Assigning variables & rules for variable names
- Types of objects we met: **integers**, **floats**, **strings**
- How to **concatenate** and **slice** strings
- Python syntax rules:
  - Spaces and white space do not matter
  - Indentation matters
- Functions we learned: `import, print, type, isinstance, float, int, str`

Syntax Rules in Python

- Python is *case sensitive*: `letsroll` ≠ `LetsRoll`
- White space does not matter (e.g., line 9 or 11 below)
- Indentation *does* matter — use **tab** to indent your code
- **Indexing** (we'll come back to this later)
  - Python starts indexing at 0. So if you have a list of numbers:

    `list = [ 2 , 5 , 7 , 1 , 9 , 2 ]`

    and you ask for `list[1]`, you'll get 5.

- Code that follows # is not read by the interpreter:

```
 8 # this is a commented line!
 9
10 print('this line will totally run.')
11
```

From last class: Rules for creating new variable names

Variable names should only ever contain letters, numbers, and underscores.

- ○ Do not start with a number.
- ○ No spaces in variable names.
- ○ Variable names are **case-sensitive**.
- ○ Names cannot be keywords (e.g., and, break, try).
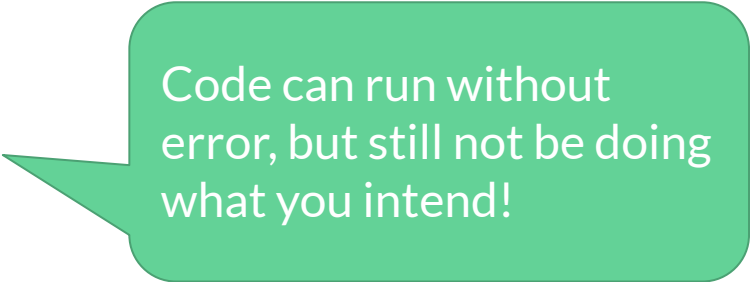- ○ Names cannot contain symbols, including dashes (-)

Troubleshooting

You'll encounter various types of errors

- **Syntax**: language rules broken
    - E.g., quotes missing, incorrect indentation
- **Runtime**: unable to execute
    - E.g., zero division error, or an unrecognized variable
- **Semantic/Logical**: unexpected output, e.g.:

```
>>> name = "Alice"
>>> print("Hello name")
>>> Hello name
```

Code can run without error, but still not be doing what you intend!

For a full list of possible errors:
https://www.tutorialsteacher.com/python/error-types-in-python

Introducing: Stack Overflow

If you have a question about something, chances are at least a thousand other people on the internet had that same question.

https://stackoverflow.com/questions/1549801/what-are-the-differences-between-type-and-isinstance

Introducing ChatGPT

[ChatGPT Coding Tutorial - A Crash Course on ChatGPT Coding for Beginners](#)

[How I'd Use ChatGPT to Learn to Code (If I Could Start Over)](#)

Be a *mover*: Make forward progress, & strike a balance between stopping & tinkering forever.

stopper

tinkerer
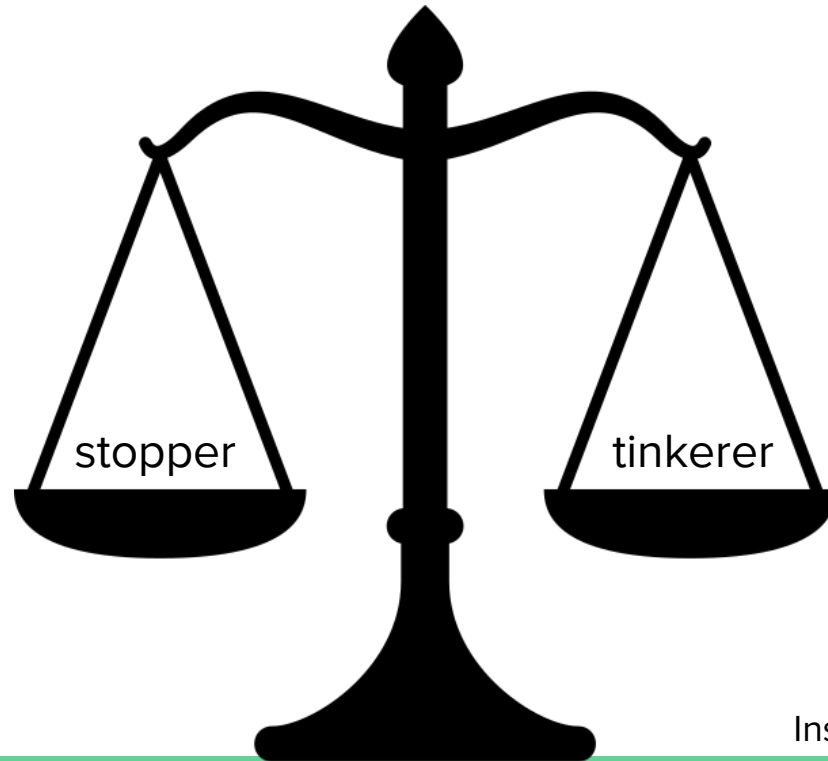
Inspired by Shannon Ellis (COGS18)

Consider the two hour rule

- If you're stuck, work on a problem for **an hour**.
- If you're still stuck, take a **30 minute break**.
- Then, try again for **30 minutes**.
- If you're still stuck, post on Canvas, or reach out to the teaching staff.

Inspired by Shannon Ellis (COGS18)

Where else can I get help?

- Canvas Discussion Boards
- Office hours
- DataQuest/Stepik Lessons
- End-of-lecture resources
- Course materials
- GitHub: programmers' social media platform
  - especially for issues related to specific codes/packages

Geek&Poke

http://geek-and-poke.com/geekandpoke/2009/7/25/the-art-of-programming-part-2.html
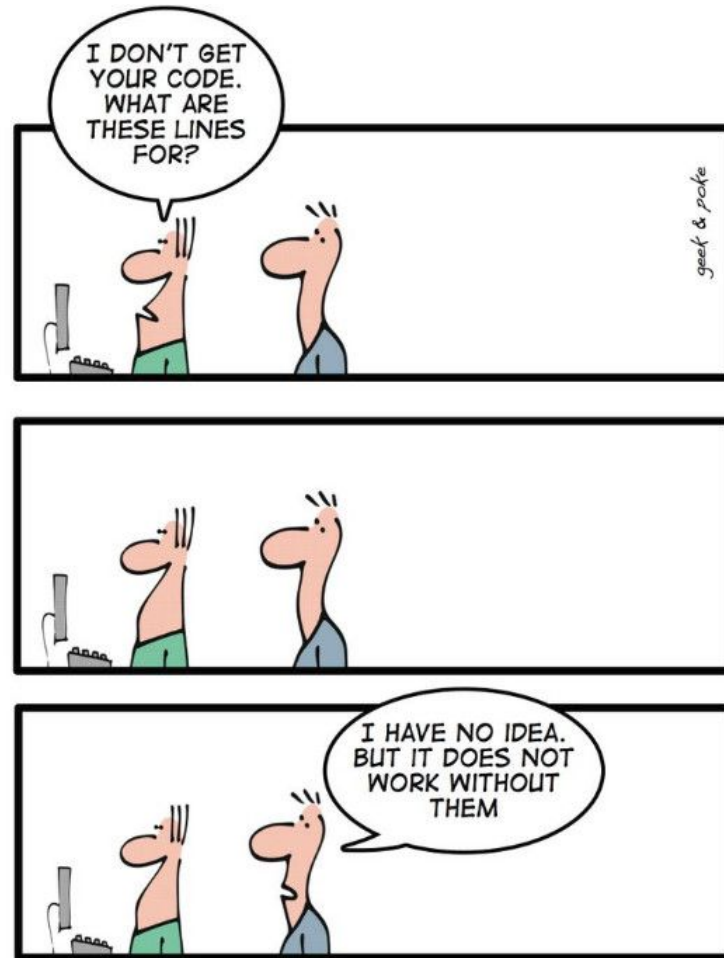
Objectives for today

- **Compare & contrast** the types of variables that Python uses to store data points
- **Understand** the syntax for lists, tuples, and dictionaries
- **Index, slice, cast, and mutate** lists
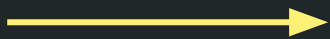
```
data_point = 8.02
```
⟶ **single variable** `(int, float, string)`
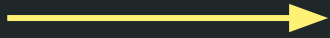
```
data = [8.38,3.34,6.35]
```
⟶ **data structure** `(list, tuple, dictionary)`
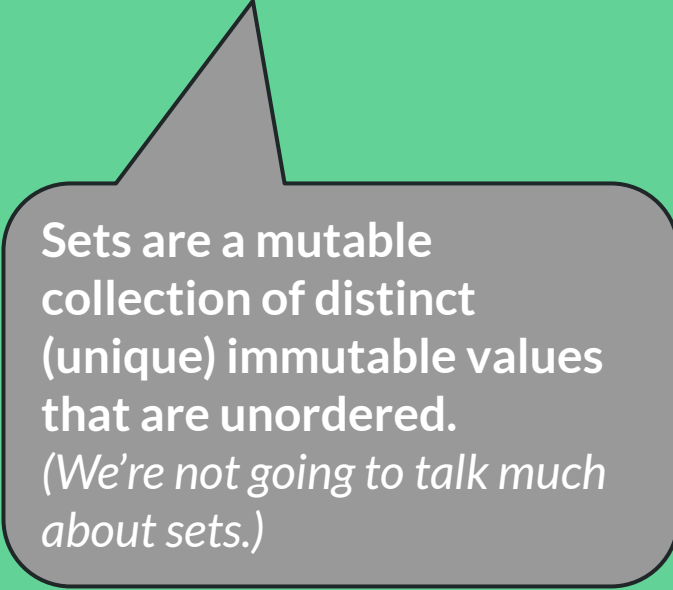
array or **dataframe**

```
big_data = [data_1,data_2,...]
```
⟶

Python has different ways to store data: **lists**, **tuples**, **dictionaries**, and **sets.**

These differ in their **syntax, mutability,** and use cases.

Sets are a mutable collection of distinct (unique) immutable values that are unordered.
(We're not going to talk much about sets.)

Which objects are immutable?

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Mutable vs Immutable Objects in Python | by megha mohan | Medium

Sets

Arrays

Dictionaries

are other types of

Tuples

Collections ← are the most popular type of — Lists

are not

are

Mutable

are created using syntax

mylist = []

are iterated by

have

Loops

often follow

Idioms

are reviewed sequentially by

are ordered groups of

Methods

such as

.sort()

such as

for elem in mylist:

Elements

.append()

are located by

which are actually

Pointers

mylist[ind]

use syntax

Indexes

can be of type

can be modified by

run forward from

run backward from

are needed for

int string list …

0 to length-1

-1 to length

Assignment

uses syntax

mylist[ind] = X

Mutable vs Immutable Objects in Python | by megha mohan | Medium

Why does Python count from 0?

Programming languages like Fortran, MATLAB and R start counting at 1 because that's what human beings have done for thousands of years!

Languages in the C family (including C++, Java, Perl, and Python) count from 0 because it represents an offset from the first value in the array (the second value is offset by one index from the first value).

This is closer to the way that computers represent arrays.

From https://swcarpentry.github.io/python-novice-inflammation/02-numpy/index.html

Lists are flexible & efficient containers for heterogeneous data

- Lists are **mutable**: we can change individual elements of the list
- Denoted by brackets & elements are separated by commas

  `my_list = [ ... , ... , ... ]`

- Check the length of your list by using `len(my list)`
- Use `my_list.append()` to add elements to a list
- Remove elements by index using `del my_list[2]`
- Remove elements by value by using `my_list.remove('oranges')`
- Sort by using `my_list.sort()`

Let's do this in the Jupyter Notebook!

Corresponding notes are here for your reference.

Lists are flexible & efficient containers for heterogeneous data

- Lists are **mutable**: we can change individual elements of the list
- Denoted by brackets & elements are separated by commas

```
my_list = ['apples', 'bananas', 'oranges']
```

- Check the length of your list by using **len(my_list)**
- Use **my_list.append()** to add elements to a list
- Remove elements by index using **del my_list[2]**
- Remove elements by value by using **my_list.remove('oranges')**
- Sort by using **my_list.sort()**

Indexing lists

```
my_list = [1,2,5,2,3]
```

Index number

```
my_list[1] = 2
```

```
my_list[-1] = 3
```

Allows you to count from the end
(could be -2, etc.)

```
my_list[5] =
```

```
IndexError
```

Shown if you try to get an index
that doesn't exist

Slicing lists

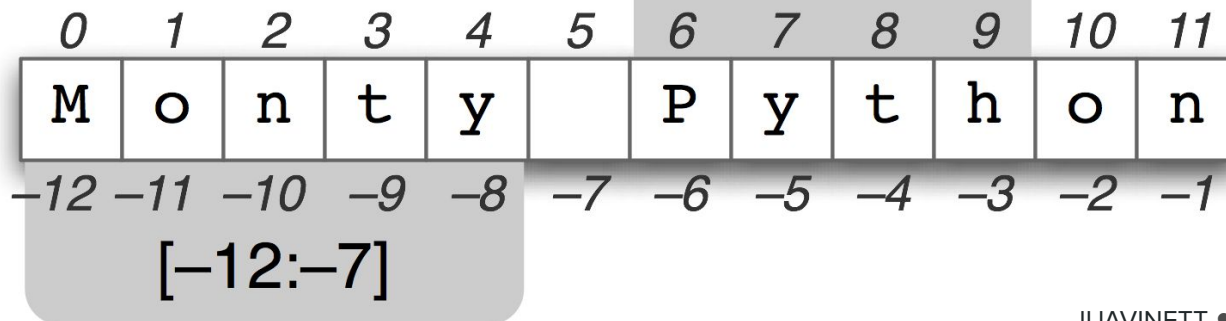`my_list[0:2]`

`my_list[1:3]`

`my_list[:3]`

`my_list[3:]`

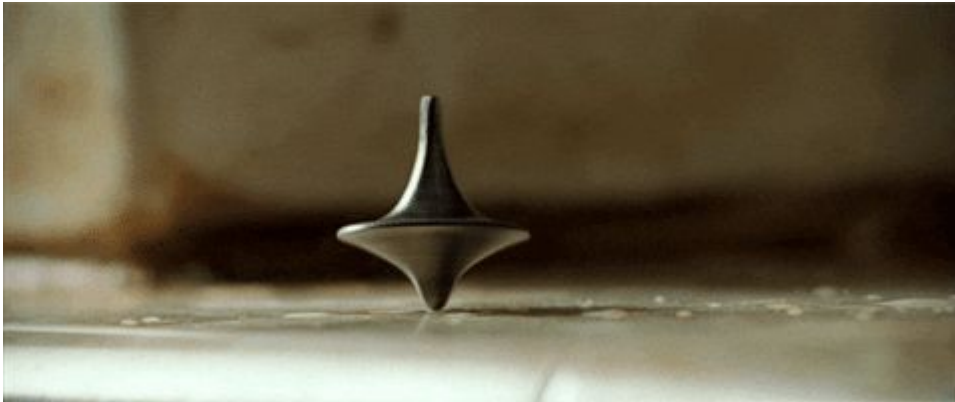`my_list[:]`

`[included:excluded]`
It doesn't show you the stop element (it shows you elements with indices 0 & 1)

One way to remember how slices work is to think of the indices as pointing between characters, with the left edge of the first character numbered 0. Then the right edge of the last character of a string of n characters has index n.

[6:10]

0	1	2	3	4	5	6	7	8	9	10	11
M	o	n	t	y		P	y	t	h	o	n

−12 −11 −10 −9 −8 −7 −6 −5 −4 −3 −2 −1

[−12:−7]

Lists of lists
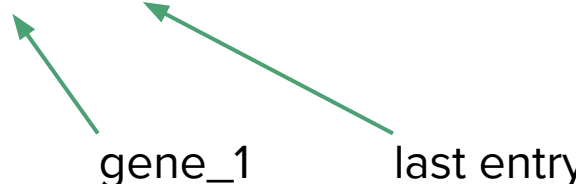
```
>>> gene_1 = ['gene1',0.48,0.55]
>>> gene_2 = ['gene2',0.38,0.85]
>>> gene_3 = ['gene3',0.21,0.81]
>>> all_genes = [gene_1, gene_2, gene_3]
>>> print(all_genes[0][-1])
```

Lists of lists

```
>>> gene_1 = ['gene1',0.48,0.55]
>>> gene_2 = ['gene2',0.38,0.85]
>>> gene_3 = ['gene3',0.21,0.81]
>>> all_genes = [gene_1, gene_2, gene_3]
>>> print(all_genes[0][-1])

>>> 0.55
```

gene_1   last entry

Tuples

- A tuple is an **immutable** collection of ordered items, that can be of mixed type.
- Tuples are created using parentheses.
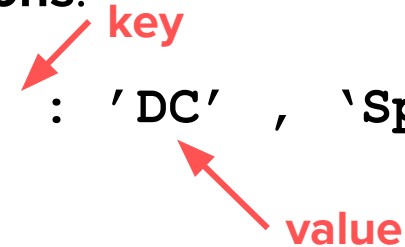- Indexing works similar to lists.

```
>>> my_tuple = ( 3, 'blue', 54.1)
```

Dictionaries link keys to values

- Denoted by **curly braces** and elements are separated by **commas**. Assignments are done using **colons**.

    key

    ```
    >>> capitals =  {'US' : 'DC' , 'Spain':'Madrid',
    'Italy:'Rome'}
    ```

    value

    ```
    >>> capitals['US']

    >>> 'DC'
    ```

- You'll get a Key Error if you ask for a key that doesn't exist
    - Use **'Germany' in capitals** to check

Working with dictionaries in Python

- Use **`capitals.update(morecapitals)`** to add another dictionary
- Use **`del capitals['US']`** to delete entries
- Loop by key or values, or both

When dictionaries are useful

1. Flexible & efficient way to associate labels with heterogeneous data
2. Use where data items have, or can be given, labels
3. Appropriate for collecting data of different kinds (e.g., name, addresses, ages)

Resources

Software Carpentries Lists

Storing Multiple Values in Lists – Programming with Python

Python 101: Lists, Tuples, and Dictionaries

Whirlwind Tour of Python: Built-In Data Structures