

# Data Structures

---

BILD 62

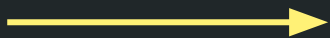
## *From last class:* Rules for creating new variable names

**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 ( - )

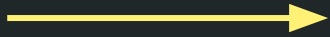
# We're learning how to deal with more and more complex data

```
data_point = 8.02
```



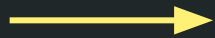
**single variable**  
(int, float,  
string)

```
data = [8.38, 3.34, 6.35]
```



**data structure**  
(list, tuple,  
dictionary)

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



**array**  
or **dataframe**



# 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
```

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>

The screenshot shows the Stack Overflow interface. The top navigation bar includes the Stack Overflow logo, links for Products, Customers, and Use cases, and a search bar. The left sidebar contains links for Home, PUBLIC, Stack Overflow (selected), Tags, Users, Jobs, TEAMS, and What's this? Below these is a badge for 'First 25 Users Free'. The main content area displays a question titled 'What are the differences between these two code fragments? Using `type()` :'. The question has 1178 votes and 446 answers. It shows two code snippets: one using `type()` and another using `isinstance()`. Below the code, there are tags for 'python', 'oop', 'inheritance', and 'types'. The question was edited by TylerH on Jan 4 '18 and asked by abbot on Oct 11 '09. It has 7 answers, with the top one by TylerH having 17.2k votes. The bottom of the screenshot shows the start of an answer by TylerH, which begins with 'To summarize the contents of other (already good!) answers, `isinstance` caters for inheritance (an instance of a derived class is an instance of a base class, too), while checking for equality of `type` does not (it demands identity of types and rejects instances of subtypes, AKA subclasses).'

stackoverflow Products Customers Use cases Search...

Home  
PUBLIC  
Stack Overflow  
Tags  
Users  
Jobs  
TEAMS What's this?  
First 25 Users Free

What are the differences between these two code fragments? Using `type()` :

1178

```
import types

if type(a) is types.DictType:
    do_something()
if type(b) in types.StringTypes:
    do_something_else()
```

Using `isinstance()` :

```
if isinstance(a, dict):
    do_something()
if isinstance(b, str) or isinstance(b, unicode):
    do_something_else()
```

python oop inheritance types

share improve this question

edited Jan 4 '18 at 22:09  
TylerH  
17.2k 10 57 75

asked Oct 11 '09 at 3:50  
abbot  
22.5k 5 43 52

add a comment

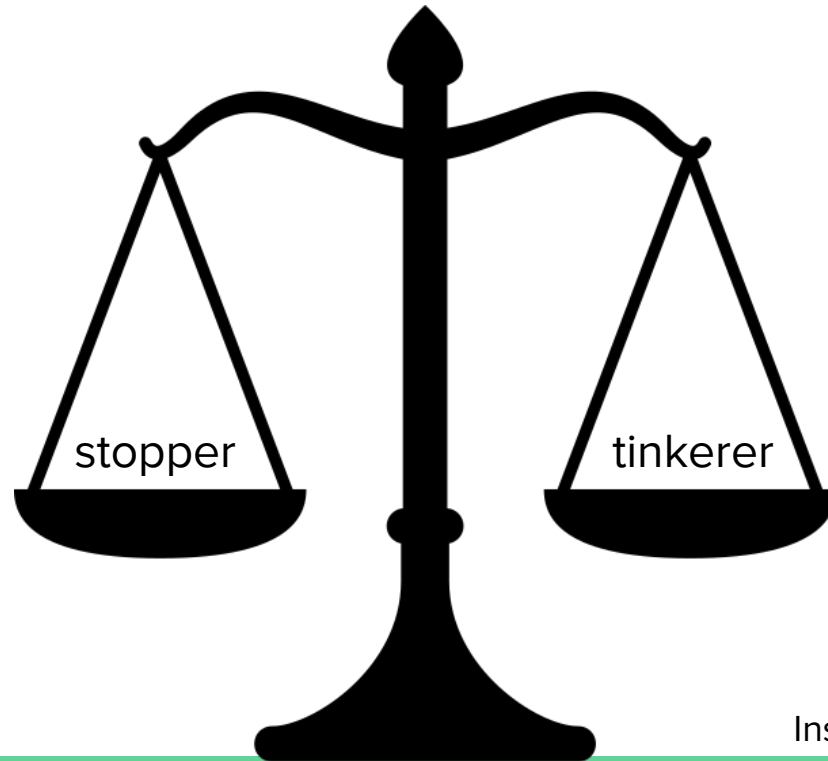
7 Answers

active oldest votes

To summarize the contents of other (already good!) answers, `isinstance` caters for inheritance (an instance of a derived class is an instance of a base class, too), while checking for equality of `type` does not (it demands identity of types and rejects instances of subtypes, AKA subclasses).

1203

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



Inspired by Shannon Ellis (COGS18)

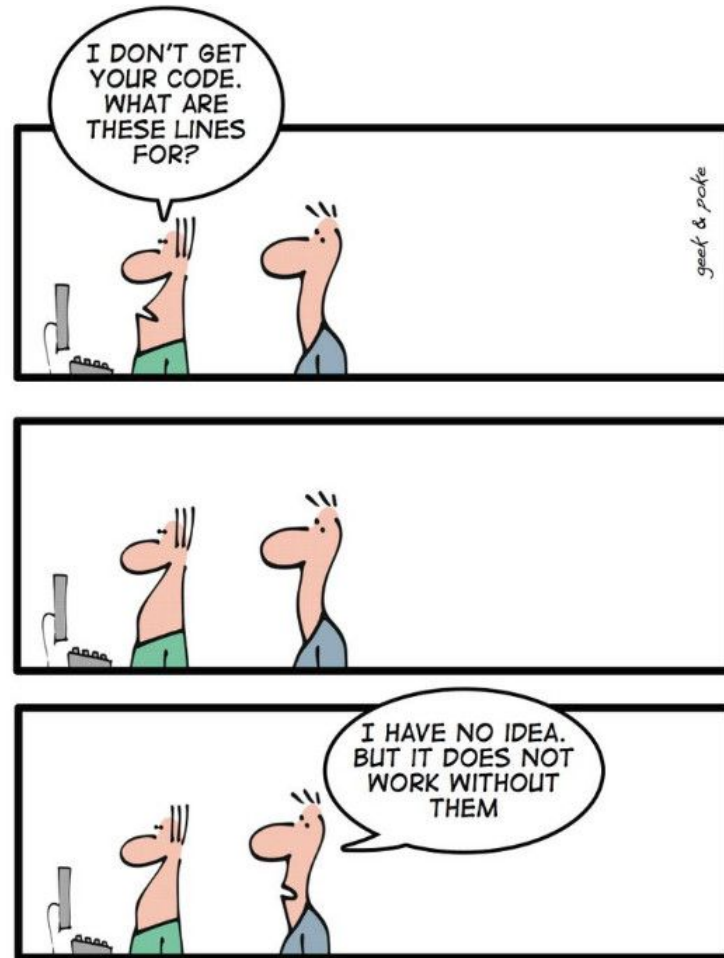
# Consider the two hour rule

- 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.



# Where else can I get help?

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



Geek&Poke

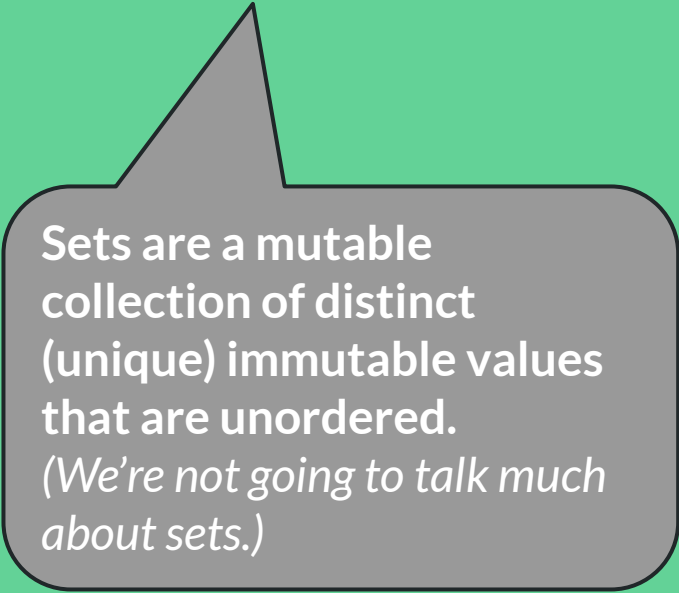
<http://geek-and-poke.com/geekandpoke/2009/7/25/the-art-of-programming-part-2.html>

# 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
-

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

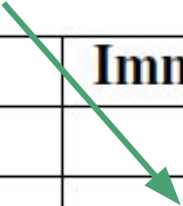
These differ in their  
**syntax, mutability,**  
and use cases.



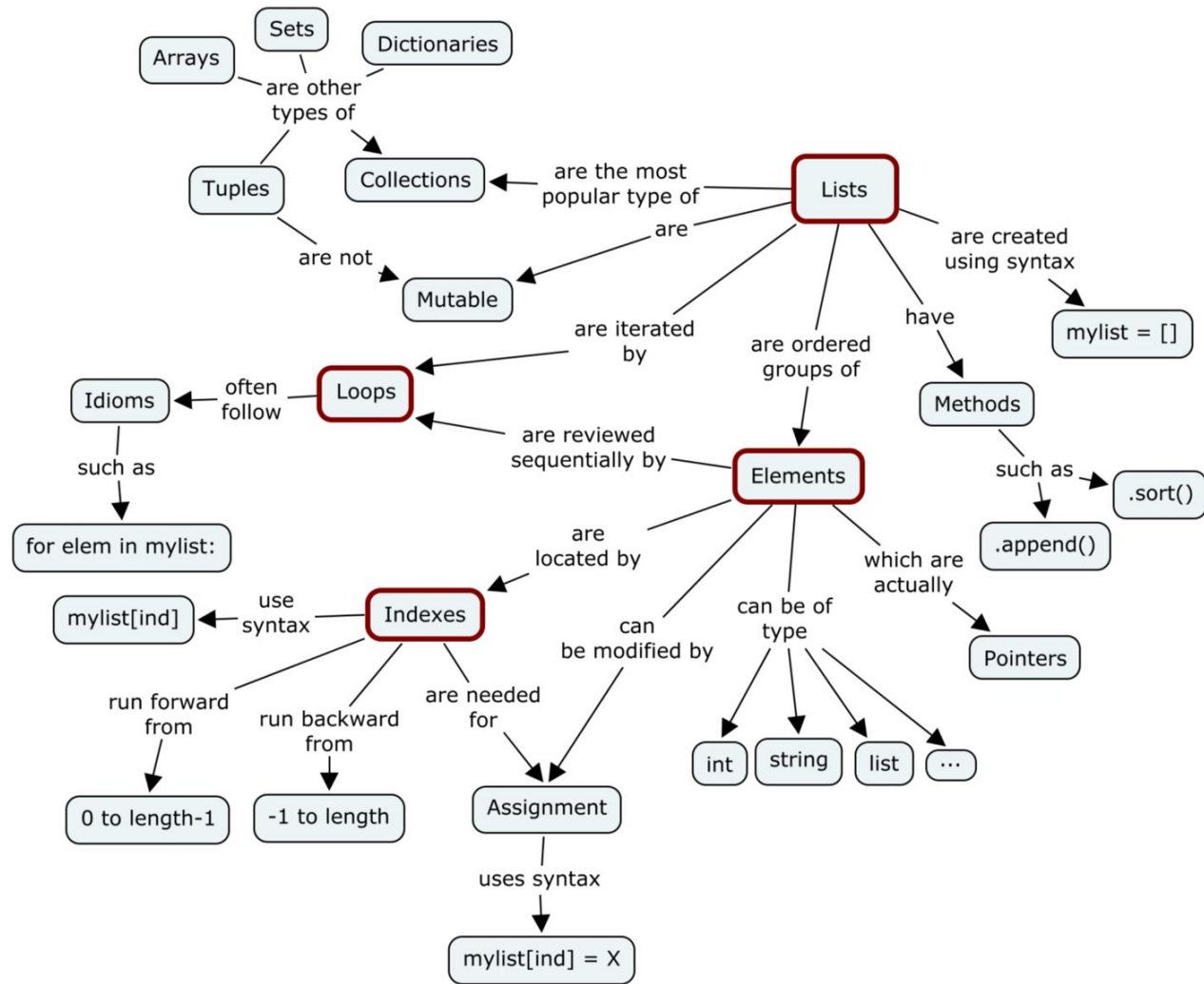
Sets are a mutable collection of distinct (unique) immutable values that are unordered.  
*(We're not going to talk much about sets.)*

# A word on mutability

Why our alias thing didn't  
"work" as expected!



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



[Mutable vs Immutable Objects in Python | by megha mohan | Medium](#)

**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']
```

Let's do this in the Jupyter Notebook!

- 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()`

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]
```

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


Index number



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

```
my_list[5] =
```

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



**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]`

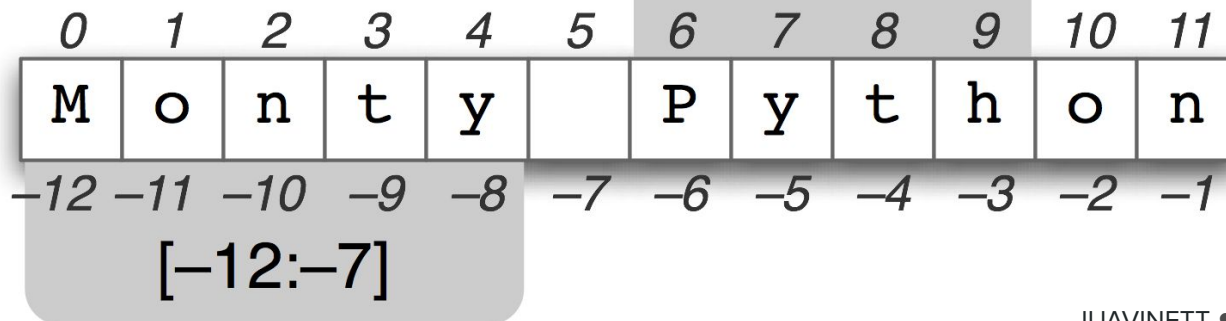
`my_list[3:]`

`my_list[:]`

**[included:excluded]**

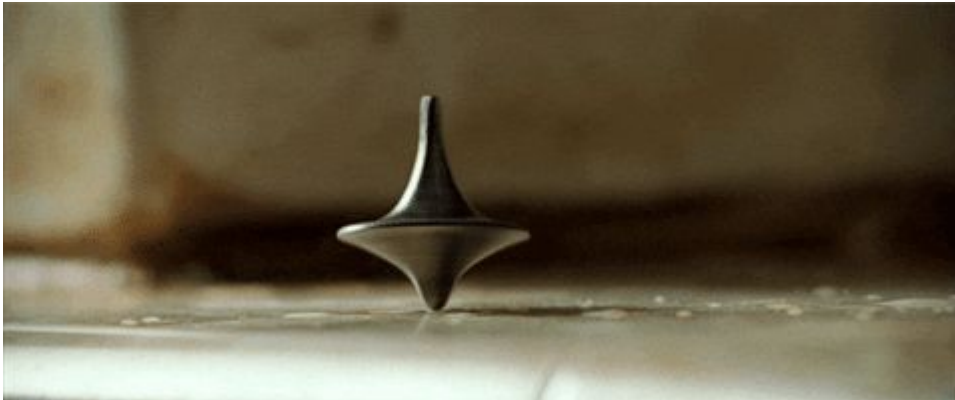
It doesn't show you the stop element (it shows you elements with indices 0 & 1)

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.



# Lists of lists

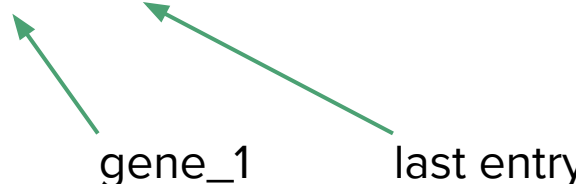
```
>>> gene_1 = ['gene1', 0.48, 0.55]
>>> gene_2 = ['gene2', 0.38, 0.85]
>>> gene_3 = ['gene3', 0.21, 0.81]
>>> all_genes = [gene_1, gene_2, gene_3]
>>> print(all_genes[0][-1])
```



# 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
```



The diagram illustrates the indexing process for the last element of the first list in a list of lists. Two green arrows originate from the text labels 'gene\_1' and 'last entry' at the bottom. The 'gene\_1' arrow points to the first element of the 'all\_genes' list, which is the list ['gene1', 0.48, 0.55]. The 'last entry' arrow points to the last element of this inner list, which is the value 0.55.

# Tuples

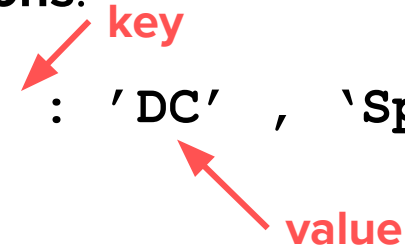
- A tuple is an **immutable** collection of ordered items, that can be of mixed type.
- Tuples are created using parentheses.
- Indexing works similar to lists.

```
>>> 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**.

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



A diagram with two red arrows. One arrow points from the word 'key' to the string 'US' in the dictionary assignment. The other arrow points from the word 'value' to the string 'DC'.

```
>>> 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](#)

[Python 101: Lists, Tuples, and Dictionaries](#)

[Whirlwind Tour of Python: Built-In Data Structures](#)