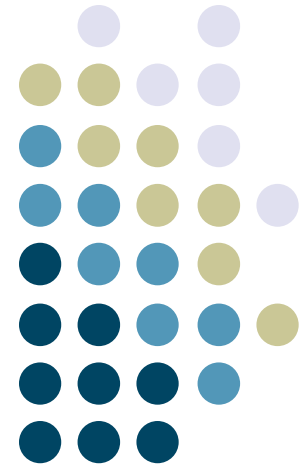


# Python Tips & Review



John Stasko

CS 6452

Prototyping Interactive Systems

# Learning Objectives

- Review some ideas
- See a new one or two
- Get some help on the HW



# Documentation



USEFUL LINKS - CS6452Fal...  
cs6452.weebly.com/useful-links.html

HOME CALENDAR ASSIGNMENTS GRADING USEFUL LINKS

CS6452 Fall-2016 USEFUL LINKS

**Python**

- Python Download
- Python 3 documentation - Library reference, tutorial, etc.
- Python for Informatics - Book from C. Severance
- Intro. to Computing using Python - Book from L. Perkovic
- Learn Python - from CodeAcademy
- Python Practice Problems - from Python Biblioteca
- New version changes - Modifications from Python 2 to 3
- Changing from Python 2 to 3 on Macs - from MacObserver
- Python 3 Cheat Sheet

**Java**

CREATE A FREE WEBSITE

USEFUL LINKS - CS6452Fal... Overview — Python 3.5.2 ...  
Python Software Foundation (US) | https://docs.python.org/3/

Python » 3.5.2 Documentation » modules | index

**Download**  
Download these documents

**Docs for other versions**  
Python 2.7 (stable)  
Python 3.4 (stable)  
Old versions

**Other resources**  
PEP Index  
Beginner's Guide  
Book List  
Audio/Visual Talks

**Quick search**  
Enter search terms or a module, class or function name.

**Python 3.5.2 documentation**

Welcome! This is the documentation for Python 3.5.2, last updated Sep 06, 2016.

**Parts of the documentation:**

- What's new in Python 3.5?**  
*or all "What's new" documents since 2.0*
- Installing Python Modules**  
*installing from the Python Package Index & other sources*
- Distributing Python Modules**  
*publishing modules for installation by others*
- Extending and Embedding**  
*tutorial for C/C++ programmers*
- Python/C API**  
*reference for C/C++ programmers*
- FAQs**  
*frequently asked questions (with answers!)*

**Indices and tables:**

- Global Module Index**  
*quick access to all modules*
- General Index**  
*all functions, classes, terms*
- Glossary**  
*the most important terms explained*
- Search page**  
*search this documentation*
- Complete Table of Contents**  
*lists all sections and subsections*

# Standard Library



The screenshot shows a web browser window displaying the Python Standard Library documentation. The browser's address bar shows the URL `https://docs.python.org/3/library/index.html`. The page title is "The Python Standard Library". On the left side, there is a navigation sidebar with sections for "Previous topic" (10. Full Grammar specification), "Next topic" (1. Introduction), and "This Page" (Report a Bug, Show Source). The main content area contains the following text:

## The Python Standard Library

While [The Python Language Reference](#) describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the [Python Package Index](#).

- 1. Introduction
- 2. Built-in Functions
- 3. Built-in Constants
  - 3.1. Constants added by the `site` module
- 4. Built-in Types
  - 4.1. Truth Value Testing
  - 4.2. Boolean Operations — `and`, `or`, `not`
  - 4.3. Comparisons
  - 4.4. Numeric Types — `int`, `float`, `complex`
  - 4.5. Iterator Types
  - 4.6. Sequence Types — `list`, `tuple`, `range`
  - 4.7. Text Sequence Type — `str`
  - 4.8. Binary Sequence Types — `bytes`, `bytearray`, `memoryview`
  - 4.9. Set Types — `set`, `frozenset`
  - 4.10. Mapping Types — `dict`
  - 4.11. Context Manager Types
  - 4.12. Other Built-in Types
  - 4.13. Special Attributes
- 5. Built-in Exceptions
  - 5.1. Base classes

# Built-in Functions



USEFUL LINKS - CS6452Fal... 2. Built-in Functions — Py... +

Python Software Foundation (US) https://docs.python.org/3/library/functions.html

Python » 3.5.2 » Documentation » The Python Standard Library »

## 2. Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

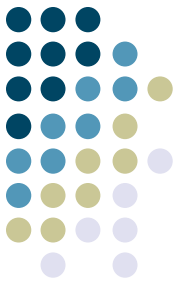
Built-in Functions				
<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

**abs(x)**  
Return the absolute value of a number. The argument may be an integer or a floating point number. If the argument is a complex number, its magnitude is returned.

**all(iterable)**  
Return `True` if all elements of the *iterable* are true (or if the iterable is empty). Equivalent to:

```
def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True
```

# Design Advice



- Break program into set of "things" that need to be done
- Make each of those things be a function
- HW?
  
- Challenge?

# Importing Data



- How would you do it?
- Recall

readCSVdictionary.py

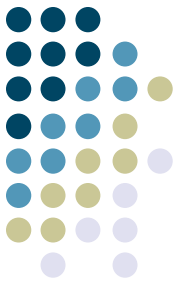
```
import csv

reader = csv.DictReader(open("students.csv"))
# puts headers in reader.fieldnames

for row in reader:
    # do work
```

What is in "reader" now?

# Back to that Challenge



- What do we have to do with "reader"?



# Data Structure



```
[  
  { "Name": "Glenmorangie", "Rating": "94", "Age": "15", ... }  
  { "Name": "Balvenie", "Rating": "87", "Age": "6", ... }  
  { "Name": "Woodford", "Rating": "90", "Age": "12", ... }  
  { "Name": "Glenfiddich", "Rating": "98", "Age": "35", ... }  
  ...  
]
```

# Match by Name

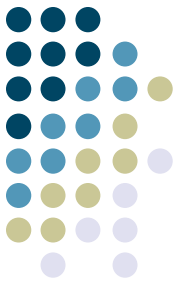


- How?
  - Walk list
  - Check name of each one
  - If match, you got it

How to check name of each one?

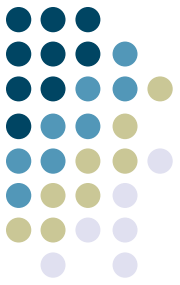
```
if target_name == row["Name"]:
```

# Matching Prices



- How?
- Walk list, check each price in range
- Then what? (sorted by rating)
- It's OK to create other useful data structures

# Formula



- Need to calculate formula for each whiskey
  - What if a bad rating or price?
- Next steps
  - Sort all "valid" whiskeys by factor
  - Produce the top 10

# Useful Function



## `sorted(iterable[, key][, reverse])`

Return a new sorted list from the items in *iterable*.

Has two optional arguments which must be specified as keyword arguments.

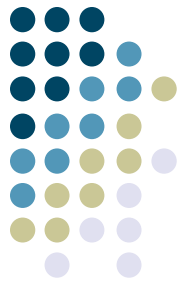
*key* specifies a function of one argument that is used to extract a comparison key from each list element: `key=str.lower`. The default value is `None` (compare the elements directly).

*reverse* is a boolean value. If set to `True`, then the list elements are sorted as if each comparison were reversed.

Use `functools.cmp_to_key()` to convert an old-style *cmp* function to a *key* function.

The built-in `sorted()` function is guaranteed to be stable. A sort is stable if it guarantees not to change the relative order of elements that compare equal — this is helpful for sorting in multiple passes (for example, sort by department, then by salary grade).

For sorting examples and a brief sorting tutorial, see [Sorting HOW TO](#).



```
>> a = [5, 8, 3]
>> b = sorted(a)
>> print(b)
[3, 5, 8]
```

What's our challenge?



```
def getFactor(obj):  
    return obj["FactorValue"]
```

```
sorted_list = sorted(factor_list, key=getFactor, reverse=True)
```

# Learning Objectives

- Review some ideas
- See a new one or two
- Get some help on the HW



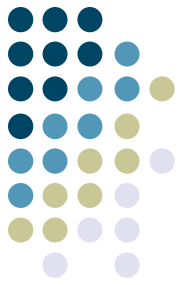


# HW 2



- Due Thursday at class time
- Ask questions in Piazza
- Sakshi and I have office hours tomorrow

# Next Time



- Accessing data files from the web
- Using web APIs