Administrivia

- Homework 7 due on Friday
- On RELATE, not Codelab
- Test your code before submitting it
- Install Anaconda

https://www.continuum.io/downloads
REVIEW
s="ABACAB"

d=

for a,b in enumerate(s):
    if b not in d:
        d[b]=0
    d[b]+=a

da) d={1:"A",3:"C","2":"B"}
b) d={"A":3,"B":2,"C":1}
c) d={"A":6,"B":6,"C":3}
d) d={"A":0,"B":0,"C":0}
s=“ABACAB”
d={}
for a, b in enumerate(s):
    if b not in d:
        d[b] = a
    else:
        d[b] += a
MODULES
Modules

• A collection of Python specialized functions, variables, and types
• We need to `import` the module
  ```python
  import math
  ```
• Can then access things within the module using `attribute operator`
  ```python
  math.sqrt(math.pi)
  ```
From

• Can choose what to import with `from`
  ```python
define3_math_special_functions
```  ```python
from cmath import phase
phase(1+1j)
```
• We don’t have to type the module name all the time
• Import multiple items with a comma
  ```python
from cmath import phase, rect
```
math.degrees(2*pi)

What should replace the ???

a) from math import pi
   import math

b) from math import pi, degrees

c) import pi
   import math

d) import math
What should replace the ???

a) from math import pi
   import math

b) from math import pi, exp

c) import pi, exp
   import math

d) import math
Useful Python Modules

- math, cmath
- random
- csv
- sys, os
- time, datetime
- itertools
- logging

- NumPy
- SciPy
- matplotlib
WRITING CODE
Writing readable code

• We should always strive to write code that is easy to read.
  – Our variables should have *descriptive* names.
  – We should also *annotate* our code.

• **REMEMBER**: A program is set of instructions a computer executes *to achieve a goal.*
Commenting

• **Comments** are text that the interpreter ignores
• Comments help *a person* read a program
• The # symbol indicates a comment
  – Anything after that symbol is ignored

# Hello, I am a comment
Docstring

- A string literal that behaves like a comment
- Use triple quotes
- Especially useful after function definition

"""Hello, I am a docstring."""

• A string literal that behaves like a comment
• Use triple quotes
• Especially useful after function definition

"""Hello, I am a docstring.""""
What is the final value of x?

a) ABCD
b) ABCD1
c) ABCD12
d) ABCD123
Why comment/document?

• Allows us to *explain* our code to others.
• But mostly… to ourselves.
• Yes, *ourselves.*
Documenting Modules

• Every script (.py) file you write is a module.
• Your modules should have a docstring at the beginning describing them and you.

"""
CS101 class demonstration
Author: Ryan Cunningham
"""
Documenting Functions

- Use doc string and describe what function does.
- Describe all parameters by name.
- Describe all return values.

def sqrt(n):
    """
    Computes square root of a number.
    n: an integer or float
    returns: the square root of n
    """
    return n**.5
```python
category_counts={}  # accumulator for counting categories
for line in open("jeopardy.txt"):
    if line[0]!="#":  # ignore comment lines
        split=line.split("\t")  # split on tab
        category=split[2]  # category is 3rd column
        # increment count of this category
        if category not in category_counts:
            category_counts[category]=0
        category_counts[category]+=1

# sort category counts for display
category_tuples=[]
for c in category_counts:
    n=category_counts[c]
    category_tuples.append((n,c))
category_tuples.sort()  # tuples are sorted by first item
print category_tuples
```
DEBUGGING
“My code doesn’t work.”
“My code doesn’t work.”
"My code doesn’t work."

1. How do I know it isn’t working?
2. What do I expect it to do?
3. What is my code doing instead? Why?

Eliminate all other factors, and the one which remains must be the truth.
Errors = Clues

1. Read message and **think** about it.

   **AttributeError**: 'int' object has no attribute 'append'

   **IndexError**: list index out of range

2. Google the error message

   **TypeError**: unsupported operand type(s) for %: 'int' and 'str'
Error messages

Google

TypeError: unsupported operand type(s) for %: 'int' and 'str'

About 20,000 results (0.55 seconds)

TypeError: unsupported operand type(s) for -: 'str' and 'int'
stackoverflow.com/.../typeerror-unsupported-operand-types-for-str-and-i...
Mar 4, 2010 - The reason this is failing is because (Python 3) input returns a string. To convert it to an integer, use int(some_string). You do not typically keep track ...

python - TypeError: unsupported operand type(s) for -: 'int ...
stackoverflow.com/.../typeerror-unsupported-operand-types-for-str-and-i...
Mar 6, 2012 - print i + " " + e + " = " + (1*e) TypeError: unsupported operand type(s) for -: 'int' and 'str' ... Probably because i and e are not strings? Try print i + e, '="', (1 * e) ...

python - Unsupported operand type(s) for -: 'int' and 'str ...
stackoverflow.com/questions/.../unsupported-operand-types-for-int-and-int ...
Dec 7, 2013 - num2 =int(input("What is your second number? ... TypeError: unsupported operand type(s) for -: 'int' and 'str' ... Explicit int to str conversion:
Don’t just stare at your code!

• Make the **computer** work, not you!
• Add print statements (especially in loops!)

Data! Data! Data!
I can't make bricks without clay.
# Find all words whose first two letters are the same
for line in open("words.txt"):
    line = line.strip()
    if len(line) >= 2:
        a, b = line[1:3]
        if a == b:
            print(line)

ValueError: need more than 1 value to unpack
The game is afoot!
# Find all words whose first two letters are the same

```python
for line in open("words.txt"):  
    line=line.strip()  
    if len(line) >= 2:  
        print(line)  
        a, b = line[1:3]  
        print(a, b)  
        if a == b:  
            print(line)
```

It is a capital mistake to theorize in advance of the facts.
# Find all words whose first two letters are the same

for line in open("words.txt"):
    line=line.strip()
    if len(line)>=2:
        a,b=line[0:2]
        if a==b:
            print(line)

Elementary!
Debugging Tips

1. Comment out lines to narrow down your search for the problem.
2. Add print statements.
3. Trace troublesome input through your code line by line.
4. Explain the problem to someone else.
5. Take a break.
Coding tips

1. **START EARLY!**

2. Break the problem down and write functions.

3. Document your functions *before* writing them.

4. Test every function (even every *line*) you write to make sure it works (write then execute, write then execute).
Ultimate Coding Secret

• If you’re confused, frustrated, and can’t make progress…

   Always approach a case with an absolutely blank mind.

• … start over.
You know my methods. Apply them.
Main function

• Allows our module to be imported OR run from the command line as a script
• Put the “starting point” code in a function called “main”
• This test checks if running on command line:

```python
if __name__ == '__main__':
    main()
```
MATPLOTLIB MODULE
Matplotlib

- Plotting library (module) for Python
- Not included in standard Python libraries
- Pylab interface imitates Matlab

```python
import matplotlib.pyplot as plt
x = [0, 1, 2]
y = [1, 2, 3]
plt.plot(x, y)
plt.show()
```
Title and Axis Labels

• Should always label axes and title our graphs

```python
plt.title("Example plot")
plt.xlabel("X data")
plt.ylabel("Y data")
```
Plot

- Function takes at least two arguments
  - List of x and y coordinates
- Can optionally take a string indicating the color/shape of the line

```python
plt.plot(x, y, 'r.').
```
- Can also take a `label` keyword argument

```python
plt.plot(x, y, 'r.', label="Fun!")
```
Plot

- Can plot multiple lines at once

```python
x = [1, 2, 3, 4]
y1 = [2, 4, 6, 8]
y2 = [3, 6, 9, 12]
plt.plot(x, y1, 'r-', label="2x")
plt.plot(x, y2, 'g--', label="3x")
```
Legend

• With multiple lines, a legend is helpful

plt.legend()
Bar charts

- Used to show amounts associated with a set of values

```python
a=[1,2,3]
b=[3,4,4]
ticks=['Donna','Jo','Sandy']
plt.xticks(x,ticks)
plt.bar(a,b,label="Bar")
```
Exercise

• Plot the year vs. HR in baseball data for 1980-present
import csv
import matplotlib.pyplot as plt

homers={}
f=open("./lahman-csv_2015-01-24/Batting.csv")
for r in csv.DictReader(f):
    y=int(r["yearID"])
    if y<1980:
        continue
    h=r["HR"]
    h.strip()
    if h=="":
        h=0
    else:
        h=int(r["HR"])
    if y not in homers:
        homers[y]=0
    homers[y]+=h
f.close()
x=[]
y=[]
ticks=[]
for year in homers:
    hr=homers[year]
    x.append(year)
    y.append(hr)
    ticks.append(str(year))

plt.xticks(x,ticks)
plt.bar(x,y)
plt.title("Year vs. HR\nin MLB")
plt.show()
Histograms

• Used to show *distribution* of values

```python
counts=[1,1,2,3,1,2,1,3,1,2,1]
plt.hist(counts)
```

• Bins the values and counts *frequency* of values that range
Exercise

- Plot distribution of word use in Jeopardy questions
counts = {}
for line in open("jeopardy.txt"):  
    if line[0] != ":" :  
        split = line.split("\t")  
        air_date, answer, category, question,  
        rnd, show_number, value = split  
        value = value.replace("",""').strip()  
        if value == "" :  
            continue  
        words = question.lower.split()  
        for word in words :  
            if word not in counts :  
                counts[word] = 0  
            counts[word] += 1
x=[ ]
for word in counts:
    x.append(counts[word])

x.sort()
x=x[-100:]  # plot only the top 100 counts

import matplotlib.pyplot as plt
plt.hist(x)
plt.xlabel("Number of times used")
plt.ylabel("Number of words")
plt.show()
Scatter Plot

- Scatter plots graph points in 2D

```python
plt.scatter(x, y,
    label="Data",
    color='g',
    marker='*')
plt.show()
```
Exercise

• Plot the year vs. RBI for Pete Rose
import csv
import matplotlib.pyplot as plt

f = open("./lahman-csv_2015-01-24/Batting.csv")
years = []
rbis = []
for r in csv.DictReader(f):
    if r["playerID"] == "rosepe01":
        y = int(r["yearID"])
        rbi = int(r["RBI"])
        years.append(y)
        rbis.append(rbi)

f.close()

plt.scatter(years, rbis)
plt.show()}