Roadmap

Programming Basics

The Console

Images

Data structures

Midterm

Graphics

Object-Oriented Programming

Everyday Python

Life after CS106AP!

Day 1!
Today’s questions

How can I organize my data so it’s easier to use?
Today’s topics

1. Assignment 2 Takeaways
2. Review
3. Organizing Data
   - Lists 2.0
   - Dictionaries
4. What’s next?
Assignment 2 Takeaways
Anatomy of a Program

- Decomposed into separate functions
- Each function has its own tests
- Work on each function separately
def main():
    ...
    c(args[0])
    ...

def c(filename):
    ...
    b(lst)
    ...

def b(pts):
    ...
    a(x, y)
    ...

def a(x, y):
    ...
    ...
Assignment 3 FYIs
Doctests

- We write all the Ghost Doctests for you!
- Doctests are analogous running code in an interpreter
  - We write the expected output on the following line.
  - Expected output could be nothing!
Review
What’s in a text file?

- The suns are able to fall and rise:
- When that brief light has fallen for us,
- we must sleep a never ending night.

- No bold/italics!
- Each line is ended by the ‘\n’ newline character!
  - Except for the last line, which doesn’t have a ‘\n’.
Parsing

**Definition**

Parsing

The act of reading “raw” text and converting it into a more useful format stored in memory.

*Adapted from Jon Skeet*
Components of Parsing

● File Reading

● String Manipulation

● Advanced Control Flow

● Container Data Types
Boolean String Functions

s.isalpha()
s.isdigit()
s.isspace()
s.startswith(substr)
s.endswith(substr)
substr in s
Find() & Strip()

```python
>>> s = 'hello!
   '   # find() returns the index of the first occurrence of the substring you pass in
>>> s.find('!')
5

>>> s.strip()  # strip() removes whitespace on left & right sides of string
'hello!'
```
Compound Boolean Expressions + Short Circuiting

s = 'yay'

if len(s) == 2 and s[1] == 'a':
    # do something

False

This will never get executed!

Definition

Short circuiting
Stopping the evaluation of a boolean expression because the overall value has already been determined.
Compound Boolean Expressions + Short Circuited

s = 'yay'

if len(s) == 3 or s[0] == 'w':
    # do something

True

This will never get executed!

Definition

Short circuiting
Stopping the evaluation of a boolean expression because the overall value has already been determined.
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
    print(word)
Break

# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)
def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output

the
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
    print(word)

all_words = ['the', 'wheels', 'on',
             'the', 'bus', 'go', 'round', 'and',
             'round', 'round', 'and', 'round',
             'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output

the
wheels
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output
the
wheels
on
Break

# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
    print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            break
    print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output

the
wheels
on
the
bus
go
# print words in all_words that aren’t censored!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

censored_words = ['round', 'circle']
censored(all_words, censored_words)
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output

the
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output

the
wheels
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output

the wheels on
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output:
the
wheels
on
the
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

censored_words = [‘round’, ‘circle’]
censored(all_words, censored_words)

Output

the
wheels
on
the
bus
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

censored_words = ['round', 'circle']
censored(all_words, censored_words)
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)

Output
the
wheels
on
the
bus
go
and
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

all_words = ['the', 'wheels', 'on', 'the', 'bus', 'go', 'round', 'and', 'round', 'round', 'and', 'round', 'round', 'and', 'round']
censored_words = ['round', 'circle']
censored(all_words, censored_words)
# print words in all_words until hit a censored word!

def censored(all_words, censored_words):
    for word in all_words:
        if word in censored_words:
            continue
        print(word)

censored_words = ['round', 'circle']
censored(all_words, censored_words)
result = ''
at = s.find('@')

curr = at + 1
while curr < len(s) and s[curr].isalpha():
    result += s[curr]
curr += 1

print(result)
While vs. For

```python
result = ''
at = s.find('@')

curr = at + 1
while curr < len(s) and s[curr].isalpha():
    result += s[curr]
    curr += 1

print(result)
```

```python
result = ''
at = s.find('@')

for i in range(at + 1, len(s)):
    if not s[i].isalpha():
        break
    result += s[i]

print(result)
```
While vs. For

result = ''
at = s.find('@')

curr = at + 1
while curr < len(s) and s[curr].isalpha():
    result += s[curr]
curr += 1

print(result)

result = ''
at = s.find('@')

for i in range(at + 1, len(s)):
    if not s[i].isalpha():
        break
    result += s[i]

print(result)

What's the difference?
result = ''
at = s.find('@')

curr = at + 1
while curr < len(s) and s[curr].isalpha():
    result += s[curr]
curr += 1

print(result)

result = ''
at = s.find('@')

for i in range(at + 1, len(s)):
    if not s[i].isalpha():
        break
    result += s[i]

print(result)

What's the difference?
They print the same result!
While vs. For

```python
result = ''
at = s.find('@')
curr = at + 1
while curr < len(s) and s[curr].isalpha():
    result += s[curr]
curr += 1
print(result)
```

```python
result = ''
at = s.find('@')
for i in range(at + 1, len(s)):
    if not s[i].isalpha():
        break
    result += s[i]
print(result)
```

Stylistically preferred
Concatenating vs. Slicing

```python
result = ''
at = s.find('@')

curr = at + 1
while curr < len(s) and s[curr].isalpha():
    result += s[curr]
    curr += 1

print(result)
```

```python
at = s.find('@')
curr = at + 1
while curr < len(s) and s[curr].isalpha():
    curr += 1

print(s[at+1:curr])
```
Concatenating vs. Slicing

result = ''
at = s.find('@')
curr = at + 1
while curr < len(s) and s[curr].isalpha():
    result += s[curr]
curr += 1
print(result)

at = s.find('@')
curr = at + 1
while curr < len(s) and s[curr].isalpha():
    curr += 1
print(s[at+1:curr])

You can slice instead of building up a result string!
Think/Pair/Share:

Print list of zoo animals (not including the bears) and corresponding list of number of times each animal has been fed.
How can I organize my data so it’s easier to use?
Recall:

['hansa', 'kandula', 'lumpy', 'suras']

[4, 3, 2, 6]
Recall:

['hansa', 'kandula', 'lumpy', 'surus']

[4, 3, 2, 6]

These pieces of information are linked!
Recall:

['hansa', 'kandula', 'lumpy', 'suras']

[4, 3, 2, 6]

These pieces of information are linked!

Can we store them so they’re associated with each other?
Introducing...
Dictionaries!
Dictionary
A container data type that maps “keys” to their associated “values”. 

Definition
Adapted from Jon Skeet
Anatomy of a Dictionary

d = {'hansa': 4, 'kandula': 3, 'lumpy': 2, 'surus': 6}
Anatomy of a Dictionary

d = {'hansa': 4, 'kandula': 3, 'lumpy': 2, 'surus': 6}

This is a dictionary literal
Anatomy of a Dictionary

d = {'hansa': 4, 'kandula': 3, 'lumpy': 2, 'surus': 6}

This is a dictionary literal

... but it's easier to visualize it this way:
Anatomy of a Dictionary

d = {'hansa': 4, 'kandula': 3, 'lumpy': 2, 'surus': 6}

This is a dictionary literal
... but it's easier to visualize if this way:

dict

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>4</td>
</tr>
<tr>
<td>'kandula'</td>
<td>3</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>2</td>
</tr>
<tr>
<td>'surus'</td>
<td>6</td>
</tr>
</tbody>
</table>
Anatomy of a Dictionary

Each key can store one value.
Anatomy of a Dictionary - Get/Set

```python
>>> d['hansa']
```

```
<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>4</td>
</tr>
<tr>
<td>'kandula'</td>
<td>3</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>2</td>
</tr>
<tr>
<td>'surus'</td>
<td>6</td>
</tr>
</tbody>
</table>
```
Anatomy of a Dictionary - Get/Set

```python
>>> d["hansa"]
```

This operation is called “get”
Anatomy of a Dictionary - Get/Set

```python
>>> d['hansa']
4
```

This operation is called “get.”

```
dict
   keys         values
   'hansa'      4
   'kandula'    3
   'lumpy'      2
   'surus'      6
```
Anatomy of a Dictionary - Get/Set

```python
>>> d[‘hansa’]
4

>>> d[‘hansa’] = 5
```
Anatomy of a Dictionary - Get/Set

```python
>>> d['hansa']
4

>>> d['hansa'] = 5
```

This operation is called “set”
Anatomy of a Dictionary - Get/Set

```python
>>> d['hansa']
4

```
Anatomy of a Dictionary - Get/Set

```python
>>> d['hansa']
4
```

```python
>>> d['hansa'] = 5
```

```python
>>> d['nick']
dict
{'hansa': 5, 'kandula': 3, 'lumpy': 2, 'suras': 6}
```
Anatomy of a Dictionary - Get/Set

>>> d[‘hansa’]
4

>>> d[‘hansa’] = 5

>>> d[‘nick’]
KeyError

```
dict

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘hansa’</td>
<td>5</td>
</tr>
<tr>
<td>‘kandula’</td>
<td>3</td>
</tr>
<tr>
<td>‘lumpy’</td>
<td>2</td>
</tr>
<tr>
<td>‘surus’</td>
<td>6</td>
</tr>
</tbody>
</table>
```

keys
values
4
5
3
2
6
Anatomy of a Dictionary - Get/Set

```python
>>> d['hansa']
4
>>> d['hansa'] = 5
>>> d['nick']
```

```
{‘hansa’: 5, ‘kandula’: 3, ‘lumpy’: 2, ‘surus’: 6}
```

“get” errors if the key is not in the dict.
Dictionaries + in

```python
>>> 'hansa' in d
```

```
keys

| 'hansa' | 5 |
| 'kandula' | 3 |
| 'lumpy' | 2 |
| 'suras' | 6 |
```

```python
dict
```
Dictionaries + in

>>> 'hansa' in d

check if key is present

check if key is present
Dictionaries + **in**

```python
>>> 'hansa' in d
True
```

- **check if key is present**

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>5</td>
</tr>
<tr>
<td>'kandula'</td>
<td>3</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>2</td>
</tr>
<tr>
<td>'surus'</td>
<td>6</td>
</tr>
</tbody>
</table>
Dictionaries + in

>>> 'hansa' in d

True

>>> 'nick' not in d
Dictionaries + **in**

```python
>>> 'hansa' in d
True

>>> 'nick' not in d
True
```

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>'hansa'</td>
<td>5</td>
</tr>
<tr>
<td>'kandula'</td>
<td>3</td>
</tr>
<tr>
<td>'lumpy'</td>
<td>2</td>
</tr>
<tr>
<td>'surus'</td>
<td>6</td>
</tr>
</tbody>
</table>
Dictionaries + `in`

```python
>>> 'hansa' in d
True

>>> 'nick' not in d
True
```

Common pattern:
Check if key is present. If it is, do something.
If it isn’t, do something else.
Building a Dictionary

```python
>>> d = {}
```
Building a Dictionary

>>> d = {}  # create an empty dictionary
Building a Dictionary

```python
>>> d = {}
>>> d[‘hansa’] = 3
```
Building a Dictionary

```python
>>> d = {}

>>> d[‘hansa’] = 3
```

we can add keys using “set”!
Building a Dictionary

```python
>>> d = {}
>>> d[‘hansa’] = 3
>>> d
we can add keys using “set”!
```
Building a Dictionary

```python
>>> d = {}

>>> d[‘hansa’] = 3

>>> d

{‘hansa’: 3}
```

we can add keys using “set”!
Building a Dictionary

>>> d = {'hansa': 3}
Building a Dictionary

```python
>>> d = {'hansa': 3}
>>> d['hansa'] += 2
```
Building a Dictionary

```python
>>> d = {'hansa': 3}

>>> d['hansa'] += 2
we can get/set on the same line!
(same as d['hansa'] = d['hansa'] + 2)
```
Building a Dictionary

```python
>>> d = {'hansa': 3}
>>> d['hansa'] += 2
>>> d
```

we can get/set on the same line!

(same as `d['hansa'] = d['hansa'] + 2`)
Building a Dictionary

```python
>>> d = {'hansa': 3}
>>> d['hansa'] += 2
>>> d
{'hansa': 5}
```

we can get/set on the same line!
(same as `d['hansa'] = d['hansa'] + 2`)

Think/Pair/Share:

Make a dictionary of zoo animals (not including the bears) and the number of times each animal has been fed.
What’s next?
What’s next?

```
keys     | values
---------|-------
'hansa'    | 5
'kandula'  | 3
'lumpy'    | 2
'surus'    | 6
```