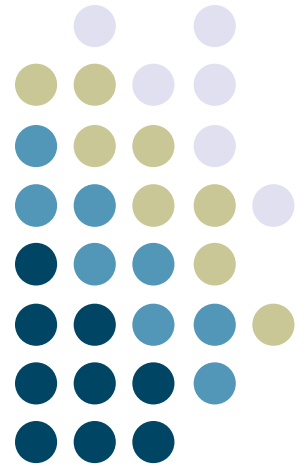


Python Strings and Data Structures



**Georgia
Tech**



John Stasko

CS 6452

Prototyping Interactive Systems

Learning Objectives



- Strings (more)
- Python data structures
 - Lists
 - Tuples
 - Dictionaries
- Get comfortable writing more code

Questions?

- Basic Python OK?
- How was the HW?



Strategies



- Don't write up your entire program all at once
- Decompose it into pieces & get each piece working independently

Multiple values



```
def mult3(a, b, c):  
    return a+1, b+2, c+3
```

```
a, b, c = mult3(1, 1, 1)
```

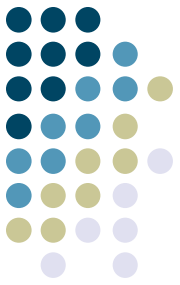
Strings



- Used everywhere
(Take out your laptops)

- ```
>>> s = "Hey!"
>>> print(s + " You")
>>> print(len(s))
>>> print(s * 3)
```

# Printing Elements



Print all the letters in a string

```
for letter in "Hello":
 print(letter)
```

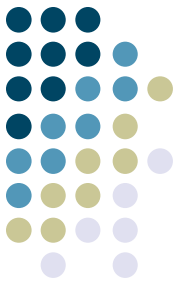
```
str = "run"
for ch in str:
 print(ch, end=' ')
```

Print only vowels?

```
str = "dictionary"
for letter in str:
 if letter in "aeiouAEIOU":
 print(letter)
```



# Reverse



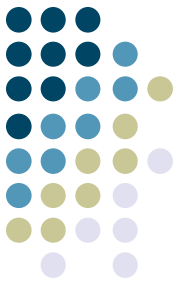
How to reverse a string?



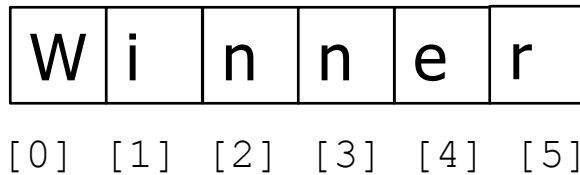
```
def reverse(str):
 result = ""
 for letter in str:
 result = letter + result
 print result
```



# Indices



Strings have indices



```
str = "Winner"
print(str[4])
```

```
print(str[-1])
print(str[-2])
```

```
print(str[6])
```

```
print(str[1:3])
```

# Alt Traversal



Traverse, print, and reverse characters with while, not for

```
def reverse2(str):
 index = 0
 rev = ""
 while index < len(str)
 print(str[index])
 rev = rev + str[index]
 print(rev)
```



**Wrong**

# Alt Traversal

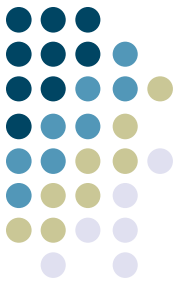


Traverse, print, and reverse characters with while, not for

```
def reverse2(str):
 index = 0
 rev = ""
 while index < len(str)
 print(str[index])
 rev = str[index] + rev
 index = index + 1
 print(rev)
```

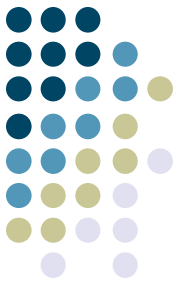


# Modify a String?



- Strings are immutable
  - Once created, cannot be changed
- So how do you “modify” one?
  
- Always create a new one

# String Operations



- Many functions on strings

`s.count(s1)` – count of how often `s1` occurs in `s`

`s.find(s1)` – Returns first index of `s1` in `s` (-1 if not there)

`s.lower()` – convert to lowercase

`s.upper()` – convert to uppercase

`s.replace(old, new)` – replaces all occurrences of `old` with `new`

`s.isalpha()` – true if only contains alphabetic characters

`s.isdigit()` – true if only numbers

`s.lstrip()` – removes leading whitespace from `s`

`s.rstrip()` – removes trailing whitespace from `s`

`s.strip()` – removes leading & trailing whitespace from `s`

`s.isupper()` – true if all uppercase

...

Remember: Some return a new string, don't modify existing one

# Useful function



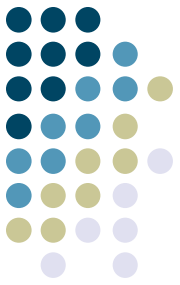
```
>>>str = " John plays golf"
>>>l = str.split()
>>>print(l)
```

```
['John', 'plays', 'golf']
```

A list (more to come soon)

```
>>> str.strip().lower().split()
???
```

# Parsing a String



Want second half of email (after @ sign) in this

From: Bruckman, Amy S asb@cc.gatech.edu Date: Fri, 26 Aug 2016 20:32:17 +0000

```
str = "From: Bruckman, Amy S asb@cc.gatech.edu Date: Fri, 26 Aug 2016 20:32:17 +0000"
pos = str.find('@')
space = str.find(' ', pos)
host = str[pos+1, space]
```



[www.faccinefb.com](http://www.faccinefb.com)

# Exercise

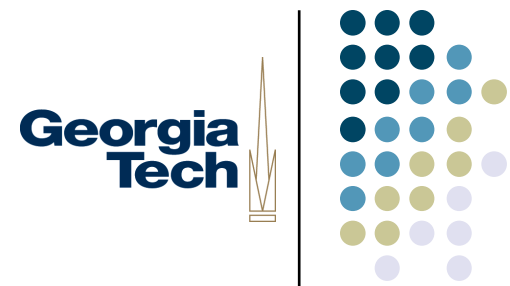


- Create a palindrome tester

```
def palindrome(str):
 start = 0
 end = len(str) - 1
 while start < end:
 if str[start] != str[end]:
 return False
 start = start + 1
 end = end - 1
 return True
```



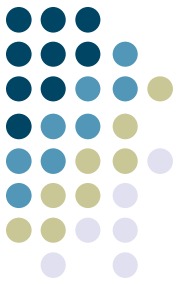
# Helpful Stuff I



- `dir` function – lists all methods on a type of object

```
>>> stuff = 'Hello world'
>>> type(stuff) <type 'str'>
>>> dir(stuff) ['capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
'expandtabs', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdigit', 'islower',
'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'partition',
'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip', 'split',
'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
```

# Helpful Stuff 2



- `help` function tells what a method does

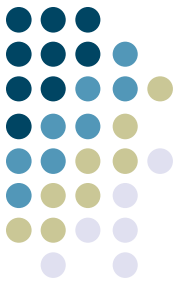
```
>>> help(str.capitalize)
Help on method_descriptor:
capitalize(...)
S.capitalize() -> string
Return a copy of the string S with only its first character capitalized.
```

# Admin Intermission

- Survey
- Piazza
- Office hours
- Slides
- Code in t-square



# Data Structures



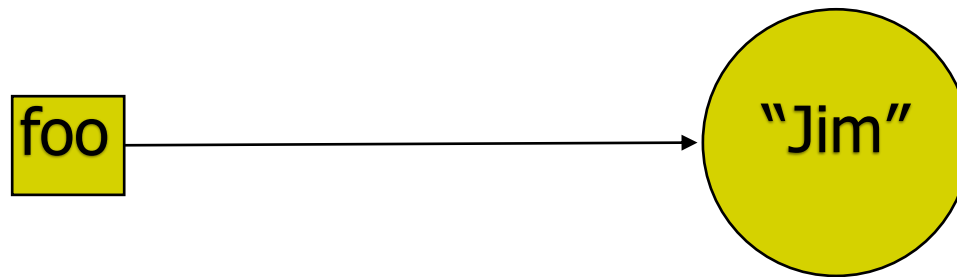
- Sometimes, you need more than a variable



# Variables

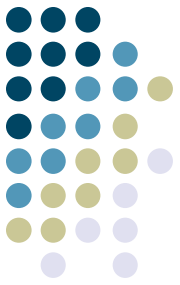


- A variable is simply a name that contains a reference to some information
- `foo = "Jim"`



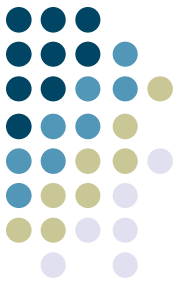
- Variables can be reassigned, and multiple variables can refer to the same thing
- Stashing a reference in a variable gives you a way to name it, and get at it later

# Problem

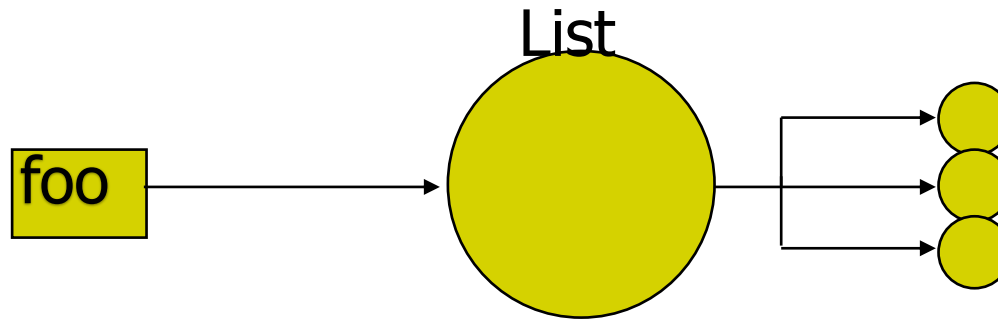


- Some more complex structures are hard to represent by just a named variable though
- Example: you want to keep track of all of the users in a chat
  - user1 = "Steven"
  - user2 = "Amy"
  - ...
- This is too *static*. Would you just create 1000 variables in case you ever had that many users? How would you do something to each one (can't easily iterate)

# Lists to the Rescue

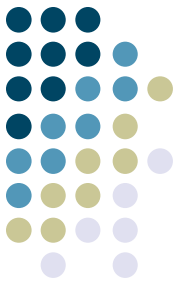


- Fortunately, Python has a built-in way to do this: *lists*
- `foo = [ "one", "two", "three" ]`



- Lists collect multiple references to data items into a single data structure
- These references are *ordered*
- The contents of the list can be altered (it is *mutable*)
- `currentChatUsers = [ "Amy", "Steven", ... ]`

# List



- Sequence of values
- Heterogeneous (not all same type of value)
- Mutable!
- Denoted with [ ]

```
[50, 40, 30, 'Mary', 'Fred']
```





```
evens = [2, 4, 6, 8]
names = ["Jim", "Jane", "Mike", "Mary"]
vals = range(5)
vals is [0, 1, 2, 3, 4]
nums = range(1,10,3)
???
for i in nums:
 print(i)
```

# Accessing Elements



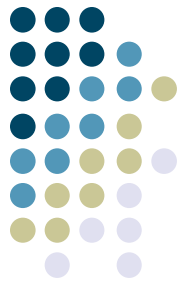
- [ ] used to get an index

```
days = ['sun', 'mon', 'tue', 'wed', 'thu', 'fri', 'sat']
c = days[3]
print(c)
print(days[-1])
week = days[1:6]
print(week)
```

```
days[2] = 'sleep'
What happens?
```

**Mutable**

# List Methods



`append(item)` – Adds item to end of list

`count(item)` – Returns count of how often item appears

`index(item)` – Returns index of first element with value item

`insert(index, item)` – Put item into list at position index and  
**slide all others over one to the right**

`sort()` – Sort items so they appear in ascending order

`remove(item)` – Remove first occurrence of item

`reverse()` – Reverses order of list

```
>>>l = ['a', 'b', 'c']
```

```
>>> del l[1]
```

```
>>> print(l)
```

# Aliases



```
list1 = [1, 2, 3, 4]
list2 = list1
list1[2] = 12
```

```
print(list1)
print(list2)
```

```
list3 = [] + list2
list3.append(10)
```

# Tuple



- Like lists, only *immutable*
  - The set of references in a tuple is fixed

- Generally used either when:

- You need a constant list

```
daysOfWeek = ("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")
```

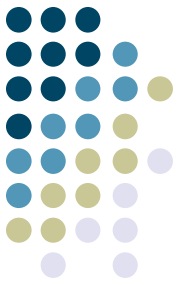
- You need to group together a set of data whose *structure* is fixed:

E.g., using tuples as quick-and-dirty records, such as address book entries:

```
myContactInfo = ("John Stasko", "TSRB355", "stasko@cc.gatech.edu")
```

- All list operations work on tuples, except ones that modify the set of references within the tuple
  - So, no `append()`, `remove()`, etc.

# Tuple



- Immutable!
- Lists of comma separated values

```
t1 = 'a', 'b', 'c'
t2 = ('a', 'b', 'c')
equivalent

t3 = tuple('bobcat')
print(t3)
t4 = (10, 20, 30, 40)
print(t4[2])
print(t4[0:2])
```

# Access



```
>>> m = ['go', 'fish']
>>> (x, y) = m
>>> x
'go'
>>> y
'fish'
>>>
```

```
>>> b, a = a, b
```

 What does that do?

# Multiple values



```
def mult3(a, b, c):
 return a+1, b+2, c+3
```

```
a, b, c = mult3(1, 1, 1)
```

## Recall



# Associating Data Items

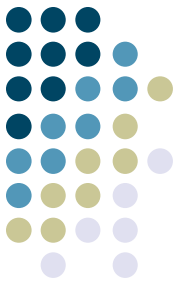


- Sometimes, you need to associate one item with another one
  - Example: hours worked on each day of the week:

|          |     |
|----------|-----|
| "Sunday" | 4.5 |
| "Monday" | 8   |
| ...      | ... |

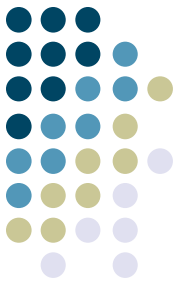
- You could do this with variables, as long as there's a fixed set of them:
  - `sunday=4.5`
  - `monday=8`

# Associating Data Items



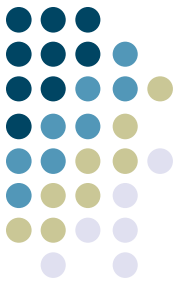
- If you don't know the associations you might have up front, you could use parallel lists:
  - `workDates = [ "1/29/05", "1/30/05", "2/1/05", ... ]`
  - `workHours = [ 4.5, 8, 5.5, ... ]`
- Then, iterate through the first list to find the date you're looking for, then look for the item with the corresponding index in the second list
- Too much work! Too error prone!
- Fortunately, Python has a built-in data structure for creating associations: the *dictionary*

# Dictionary



- Like a list, but the index can be anything
  - You state what it is
  - Called a key
- Made up of key,value pairs
- Used to store and subsequently access data
- Similar to a hash table

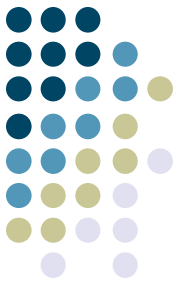
# Example



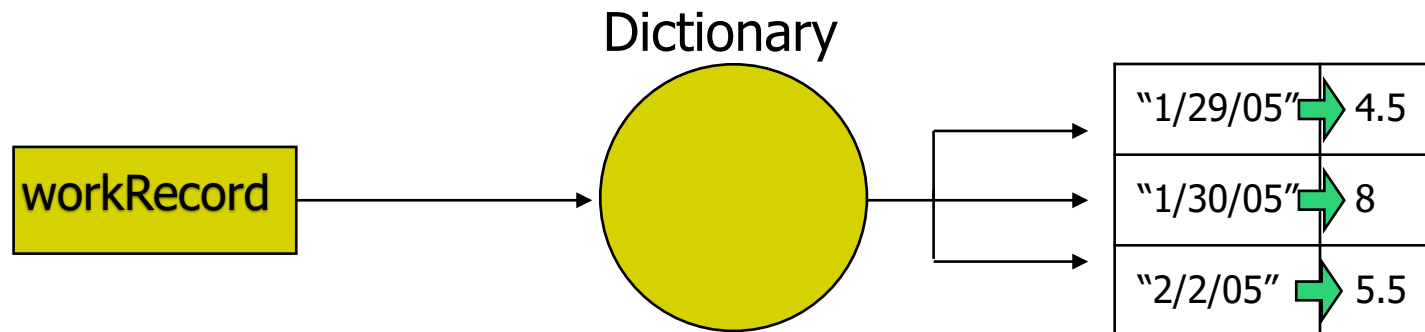
902634854, "Sally Wilson"  
917365643, "Lars Jonsson"  
931967385, "Sakshi Gupta"  
923438961, "Jiang Xiao"  
...

**Syntax:** `{ key1:val1, key2:val2, ... }`

# Dictionary Data Structure



- Dictionaries associate *values* with *keys* (you *lookup* a value given its key)
- Both are references to data items
- `workRecord = {"1/29/05":4.5, "1/30/05":8, "2/2/05":5.5 }`



- Dictionaries are the most commonly used Python data structure
- Virtually any Python data types can be used as a key or value

# Code Example

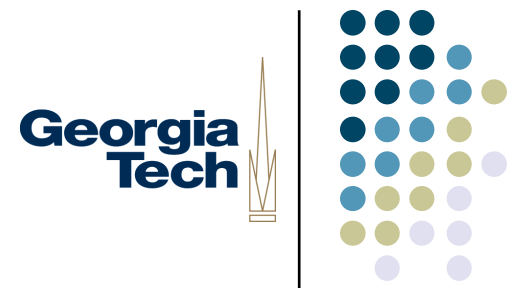


```
months = { 'Jan':1, 'Feb':2, 'Mar':3, 1:'Jan', 2:'Feb', 3:'Mar' }

print(months[2])
print(months['Jan'])

print(months.keys())
print(months.values())
```

# Important Note

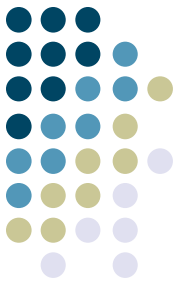


- It is not ordered, ie, order is unpredictable

```
print (months)
```

- What happens?

# Walking through



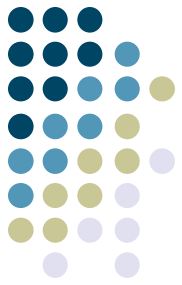
```
total = {'dave':83, 'sue':91, 'audrey':77}
for key in total:
 print(key, total[key])
```

How might you print them in sorted (alpha) order?

```
total = {'dave':83, 'sue':91, 'audrey':77}
print(total)
lst = list(total.keys())
lst.sort()
for key in lst:
 print(key, total[key])
```



# Exercise



- Want to write a program that, given a big string, counts how often each letter appears
- How do it?

# Solutions



- 1. Make 26 variables
  - Yuk
- 2. Make a list
  - Need numeric index
- Take ordinal value of character as index

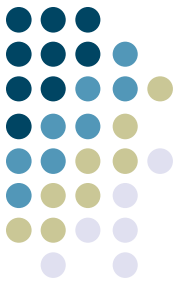
# Solutions



- 3. Use dictionary

```
word = 'areallongword'
d = dict()
for c in word:
 if c not in d:
 d[c] = 1
 else:
 d[c] = d[c] + 1
print(d)
```

# Operations Summary



`d[k]` – returns item in `d` with key `k`  
`len(d)` – returns number of items in `d`  
`list(d.keys())` – returns a list containing the keys in `d`  
`list(d.values())` – returns a list containing the values in `d`  
`k in d` – returns true if key `k` is in `d`  
`del d[k]` – removes the key `k` from `d`  
`d.get(k, v)` – returns `d[k]` if `k` is in `d`, and `v` otherwise  
`d[k] = v` – associates value `v` with key `k` in `d`  
                    (replaces an existing value, if present)  
`for k in d` – iterates over keys in `d`  
`d.items()` – returns a list of (key,value) tuples  
...

# Learning Objectives



- Strings (more)
- Python data structures
  - Lists
  - Tuples
  - Dictionaries
- Get comfortable writing more code

# Next Time

- Manipulating files
  - Reading and writing
- Starting to work with data

