PYTHON 2.7 CONCEPTS
About These Slides

These slides serves as a “one stop shop” for Python concepts we covered in Bio131.

The slides assume that you are using a lab computer (e.g., a Mac in ETC208).

These slides assume that you are programming in Python2.7. However, much of the content is the same as for Python3.

The Terminal

Python 2.7.10

To exit the Python interpreter, type

`quit()` or `exit()`
Python as a Calculator

Open Terminal, then type python in the window prompt

>>> 4+2
>>> 4-2
>>> 4*2
>>> 4/2
>>> 4*4+2
>>> 4*(4+2)

Def: An expression is an input that Python evaluates and returns as output.

Some operators are computed before others.

Parentheses can change this order.
Python as a Calculator

An assignment is an input that Python evaluates and stores in memory.

In Python:

```python
>>> A1 = 5
>>> A2 = A1 + 1
>>> newVar = A1 + A2
```

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>5</td>
</tr>
<tr>
<td>A2</td>
<td>6</td>
</tr>
<tr>
<td>newVar</td>
<td>11</td>
</tr>
</tbody>
</table>

variable (left side)  

expression (right side)
Assignments vs. Expressions

**Assignments** store a value in memory.

```python
>>> C4 = 4
>>> X = 20
>>> newVar = X/C4
```

**Expressions** output a value to the shell.

```python
>>> 10**2
>>> X/C4
>>> (9-1)*-2
```
Numerical Types

2. Evaluate the following expressions.

```>>> 4/4
>>> 4/3
>>> 4/2
>>> 4/1```

The `type()` function outputs the value’s type.

```>>> type(4)
>>> type(4.0)```

For now, two numerical types: `int` and `float`

```>>> 4.0/4.0
>>> 4.0/3.0
>>> 4.0/2.0
>>> 4.0/1.0```

**Def:** An integer (int) is a number that displays with no decimal points.

**Def:** A float is a number that displays with decimal points.
Strings

A **string** is a sequence of characters surrounded by single quotes.

The `+` operator concatenates.

```python
>>> 'a'
>>> 'ab'
>>> 'Hey there'

>>> 'A' + 'B'
```

**Def:**

**General Rule:** Inputs and outputs of expressions typically have the same type.
Lists

A list is an ordered collection of items.

\[ \text{item}_1, \text{item}_2, \ldots, \text{item}_N \]

\begin{itemize}
  \item start of list
  \item elements, separated by ","
  \item end of list
\end{itemize}

>>> myList = [5,10,15]
>>> stringList = [‘hi’,’there’,’class’]
>>> mixedList = [1,’2’,’three’,4]

The ‘+’ operator concatenates.

>>> myList + stringList
Accessing Elements in a List

To get an item from a list, use the expression `newList[i]` where `i` (an integer) is an index. The index starts at 0!

```python
>>> newList = ['one', 'two', 'three', 'four']
>>> newList[0]
'one'
>>> newList[1]
'two'
>>> newList[2]
'three'
>>> newList[3]
'four'
```
Built-in Functions

A function is a reusable block of code

**functionName**

(name of the function)

zero or more arguments

The `len()` function has one argument: the variable to compute the length of.

```python
>>> len(newList)
>>> len(myList)
>>> len(sA)
>>> len(sB)
```

`len()` works on both lists and strings
## Built-in Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Arguments (Inputs)</th>
<th>Returns (Outputs)</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>quit() or exit()</td>
<td>nothing</td>
<td>-</td>
<td>quit()</td>
</tr>
<tr>
<td>type()</td>
<td>expression</td>
<td>type</td>
<td>type(‘string’)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>type(1)</td>
</tr>
<tr>
<td>len()</td>
<td>string or list</td>
<td>integer</td>
<td>len([1,2,3])</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>len(‘string’)</td>
</tr>
</tbody>
</table>
Different ways to run Python Code

Python Interpreter

- Enter lines one by one at the prompt
- Expression outputs are **printed** to the screen.

```python
>>> x = 3+5
>>> x
8
```

Python Program

- Python evaluates each line in the file
- Expression outputs are **not printed** to the screen.
- Instead, anything that is printed to the screen requires a **print statement**

```
x = 3+5
print x
```

```
$ python example.py
8
```
Python Programs

- All Python programs must end with .py
  - example.py
  - myFile.py
  - blah.py

- Your editor should highlight words, characters, etc.

- To run a program, type \texttt{python \textless\textless\textless\textless\textless\textlangle\textfilename\textrangle.py\textrangle}\texttt{>}
  - \textless\textless\textless\textless\textless\textlangle\textfilename\textrangle.py\textrangle is the name of the file you want to run
  - The program must be in the same directory as YOU are. When you type \texttt{ls} in the Terminal, you should see the file.

\textbf{Def:} Comments are lines that are ignored by Python. They begin with a pound sign (\#)
Print Statements

Def: `print` statements explicitly write the output to the shell.

Python 2 vs. Python 3: `print` is a statement vs. a function!

```python
x = 3+5
print x
print 'Hello World!
print 'hi' + ' ' + 'there'
print 1.0
print 1+10
print 'Number Of Seats:',24
print 'Number Of Computers:',25
print ''
```

variables

strings

ints and floats

Combos of expressions (separated by commas)

empty lines
Useful Characters for Formatting Strings

The backslash \ means *interpret the next character differently*

The forward slash / is used for directory substructure

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>Add a new line (like hitting the Return key)</td>
</tr>
<tr>
<td>\t</td>
<td>Add a tab</td>
</tr>
<tr>
<td>\’</td>
<td>Add a single quote</td>
</tr>
<tr>
<td>\</td>
<td>Add a backslash</td>
</tr>
</tbody>
</table>

print ‘This is a backslash:\\’
print ‘This is a Tab!:’\t*’
print ‘Lots of new lines:\n\n\n\n\n’
FOR Loops

FOR loops are statements that specify repeated execution (also called *iteration*).

```
example.py
for item in [1,2,3]:
    print item
```

“For each element in a list, do something.”

```
example.py
for item in [1,2,3]:
    print item
```

```
example.py (unannotated)
for item in [1,2,3]:
    print item
```

Indentation matters!
FOR Loops

```python
myList = [1,2,3]
for item in myList:
    print item
```

```python
for item in [1,2,3]:
    print item
```

Example:

```python
example.py
```

```
example.py (unannotated)
```

```python
example.py
```

```
example.py (unannotated)
```

Indentation matters!
FOR Loops

```
myList = [1,2,3]
for item in myList:
    print item
    print item,'again'
print 'Done!'
```

```
myList = [1,2,3]
for item in myList:
    print item
```

```
for item in [1,2,3]:
    print item
```

```
for item in [1,2,3]:
    print item
```

```
example.py
```

```
example.py (unannotated)
```

```
example.py
```

```
example.py (unannotated)
```
How FOR Loops Work

test = 10+2
print test

myList = [1,2,3]
print 'print statements:'
print myList[0]
print myList[1]
print myList[2]

print 'FOR Loop:'
for item in myList:
    print item
    print item,'again'

print 'Done!'
Python Types

**Def:** An integer (int) is a number that displays with no decimal points.

**Def:** A float is a number that displays with decimal points.

**Def:** A string is a sequence of characters surrounded by single quotes.

**Def:** A list is an ordered collection of items.

**Def:** A boolean can be one of two values: True or False.

Note the capitalization
## Boolean Types

### Numerical Operators (input numbers, output number)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum</td>
<td>1 + 2</td>
<td>3</td>
</tr>
<tr>
<td>Difference</td>
<td>1 - 2</td>
<td>-1</td>
</tr>
<tr>
<td>Multiply</td>
<td>1 * 2</td>
<td>2</td>
</tr>
<tr>
<td>Divide</td>
<td>1 / 2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>Power</td>
<td>1 ** 2</td>
<td>1</td>
</tr>
</tbody>
</table>

### Numerical Operators (input numbers, output Boolean)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>1 == 2</td>
<td>False</td>
</tr>
<tr>
<td>Inequality</td>
<td>1 != 2</td>
<td>True</td>
</tr>
<tr>
<td>Less Than</td>
<td>1 &lt; 2</td>
<td>True</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>1 &lt;= 2</td>
<td>True</td>
</tr>
<tr>
<td>Greater Than</td>
<td>1 &gt; 2</td>
<td>False</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>1 &gt;= 2</td>
<td>False</td>
</tr>
</tbody>
</table>
## Boolean Types

### String Operators (input strings)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum</td>
<td>‘a’ + ‘b’</td>
<td>‘ab’</td>
</tr>
<tr>
<td>Equality</td>
<td>‘a’ == ‘b’</td>
<td>False</td>
</tr>
<tr>
<td>Inequality</td>
<td>‘a’ != ‘b’</td>
<td>True</td>
</tr>
</tbody>
</table>

### Boolean Operators (input booleans, output boolean)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>True and False (4&lt;5) and (6&gt;=6)</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>Or</td>
<td>True or False (4&lt;5) or ‘a’!= ‘b’</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td></td>
<td>True</td>
</tr>
<tr>
<td>Not</td>
<td>not True</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>not (4&lt;5)</td>
<td>False</td>
</tr>
</tbody>
</table>
## The IN Function

### Boolean Operators (input booleans, output boolean)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>True and False (4&lt;5) and (6&gt;=6)</td>
<td>False True</td>
</tr>
<tr>
<td>Or</td>
<td>True or False (4&lt;5) or ‘a’!= ‘b’</td>
<td>True True</td>
</tr>
<tr>
<td>Not</td>
<td>not True not (4&lt;5)</td>
<td>False False</td>
</tr>
<tr>
<td>In</td>
<td>‘a’ in [‘a’,’b’,’c’]</td>
<td>True False</td>
</tr>
<tr>
<td></td>
<td>1 in [‘a’,’b’,’c’]</td>
<td>True False</td>
</tr>
</tbody>
</table>
Python IF Statements

x = 4
y = 10
if x < y:
    print 'x is less than y.'
else:
    print 'x >= y.'

“If some value is true, do something.”

“If some value is true, do something, otherwise do something else.”

Indentation matters!

if <boolean expression>:
    statement 1
    statement 2
    ...

if <boolean expression>:
    statement 1
    ...
else:
    statement 2
    ...

“If some value is true, do something, otherwise do something else.”
Python ELIF Statements

```
x = 4
y = 10
if x < y:
    print 'x is less than y.'
elif x == y:
    print 'x and y are equal.'
else:
    print 'x > y.'
```

```
if <boolean expression>:
    statement 1
    ...
elif <boolean expression>:
    statement 2
    ...
else:
    statement 3
    ...
```
## Built-in Functions

A **function** is a reusable block of code.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Arguments (Inputs)</th>
<th>Returns (Outputs)</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>quit() or exit()</td>
<td>nothing</td>
<td>-</td>
<td>quit()</td>
</tr>
</tbody>
</table>
| type() | expression | type | type('string')
|           |           |       | type(1) |
| len() | string or list | integer | len([1,2,3])
|       |       |       | len('string') |
| min() | list of ints and/or floats | int or float | min([1,2,3]) |
| max() | list of ints and/or floats | int or float | max([1,2,3]) |
User-Defined Functions

```python
def sing():
    print 'Happy birthday to you,`
    print 'Happy birthday to you,`
    print 'Happy birthday dear Frito,`
    print 'Happy birthday to you!'
    return

sing()
```

Indention matters!

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Arguments (Inputs)</th>
<th>Returns (Outputs)</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>sing()</td>
<td>nothing</td>
<td>nothing</td>
<td>sing()</td>
</tr>
</tbody>
</table>

Functions are like small subprograms

function name zero inputs

zero inputs return nothing
User-Defined Functions

```python
def sing(person):
    print 'Happy birthday to you,,'
    print 'Happy birthday to you,,'
    print 'Happy birthday dear ' + person + ' ,'
    print 'Happy birthday to you!'
    return

sing('Anna')
```

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Arguments (Inputs)</th>
<th>Returns (Outputs)</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>sing()</td>
<td>string</td>
<td>nothing</td>
<td>sing('Anna')</td>
</tr>
</tbody>
</table>
Another Type!

**Def:** An **integer** (int) is a number that displays with no decimal points.

**Def:** A **float** is a number that displays with decimal points.

**Def:** A **string** is a sequence of characters surrounded by single quotes.

**Def:** A **list** is an ordered collection of items.

**Def:** A **boolean** can be one of two values: True or False.

**Def:** A **file** is an object that represents, well, a file.
# Reading Files in Python

## Special Syntax

```python
fname = 'mysteryFile.txt'
```

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Arguments</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>open()</code></td>
<td>two strings:</td>
<td>File</td>
<td><code>myFile = open(fname,'r')</code></td>
</tr>
<tr>
<td></td>
<td>- File name</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>- &quot;r&quot; or &quot;w&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>read()</code></td>
<td>string</td>
<td></td>
<td><code>myFile.read()</code></td>
</tr>
<tr>
<td><code>close()</code></td>
<td></td>
<td></td>
<td><code>myFile.close()</code></td>
</tr>
</tbody>
</table>

The last two functions have a special syntax – they require a `File`:
```
myFile.read()
myFile.close()
```
Reading Files in Python

```python
fname = 'mysteryFile.txt'

## open the file
myFile = open(fname, 'r')
print myFile
print type(myFile)

## read the contents of the file
## and store it in "text" variable
text = myFile.read()

## close the file
myFile.close()

## print the text variable.
print 'Text in file:'
print text
```
Manipulating Strings

```
...## read the contents of the file
## and store it in "text" variable
text = myFile.read()
...
```

To remove whitespace (new lines, spaces, tabs) at the ends of the string:

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Arguments</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>strip()</td>
<td></td>
<td>string</td>
<td>text.strip()</td>
</tr>
</tbody>
</table>

This function has a special syntax – it requires a `String` 
ˈabcd  `.strip()`
Manipulating Strings

...%
## read the contents of the file
## and store it in "text" variable
text = myFile.read()
...

`bacadae`

Split on letter `a`

Join on letter `a`

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Arguments</th>
<th>Returns</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>split()</code></td>
<td>string</td>
<td>List of strings</td>
<td><code>text.split('a')</code></td>
</tr>
<tr>
<td></td>
<td>(delimiter)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>join()</code></td>
<td>List of strings</td>
<td>string</td>
<td><code>'a'.join(['a','b'])</code></td>
</tr>
</tbody>
</table>
**While Loops**

**Def:** A *while* loop keeps iterating until a condition is not met.

```
while <Boolean Expression>:
    <do something>
```

---

## For each element in [1,2,3], print it.
```python
for item in [1,2,3]:
    print item
```

## While loop that does the same thing.
```python
item = 3
while item >= 1:
    print item
    item = item - 1
```

---

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
</tr>
<tr>
<td>--------------</td>
</tr>
<tr>
<td>item</td>
</tr>
</tbody>
</table>
While Loops

Def: A while loop keeps iterating until a condition is not met.

```python
while <Boolean Expression>:
    <do something>
```

## For each element in [1,2,3], print it.
```python
for item in [1,2,3]:
    print item
```  
## While loop that does the same thing.
```python
item = 3
while item >= 1:
    print item
    item = item - 1
```  
<table>
<thead>
<tr>
<th>Memory</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>item</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
While Loops

**Def:** A **while** loop keeps iterating until a condition is not met.

```python
while <Boolean Expression>:
    <do something>
```

## For each element in [1,2,3], print it.
```python
for item in [1,2,3]:
    print item
```

## While loop that does the same thing.
```python
item = 3
while item >= 1:
    print item
    item = item - 1
```

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>2</td>
</tr>
</tbody>
</table>
While Loops

Def: A while loop keeps iterating until a condition is not met.

```
while <Boolean Expression>:
    <do something>
```

## For each element in [1,2,3], print it.
```
for item in [1,2,3]:
    print item
```

## While loop that does the same thing.
```
item = 3
while item >= 1:
    print item
    item = item - 1
```

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
</tr>
<tr>
<td>-------------</td>
</tr>
<tr>
<td>item</td>
</tr>
</tbody>
</table>
While Loops

Def: A while loop keeps iterating until a condition is not met.

```python
while <Boolean Expression>:
    <do something>
```

## For each element in [1,2,3], print it.
```python
for item in [1,2,3]:
    print item
```

## While loop that does the same thing.
```python
item = 3
while item >= 1:
    print item
    item = item - 1
```

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
</tr>
<tr>
<td>item</td>
</tr>
</tbody>
</table>
While Loops

**Def:** A **while** loop keeps iterating until a condition is not met.

```
while <Boolean Expression>:
    <do something>
```

```
## For each element in [1,2,3], print it.
for item in [1,2,3]:
    print item

## While loop that does the same thing.
item = 3
while item >= 1:
    print item
    item = item - 1
```

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>item</td>
</tr>
</tbody>
</table>
**While Loops**

**Def:** A **while** loop keeps iterating until a condition is not met.

```python
while <Boolean Expression>:
    <do something>
```

## For each element in [1,2,3], print it.
```python
for item in [1,2,3]:
    print item
```

## While loop that does the same thing.
```python
item = 3
while item >= 1:
    print item
    item = item - 1
```

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>item</td>
</tr>
</tbody>
</table>
**While Loops**

**Def:** A `while` loop keeps iterating until a condition is not met.

```python
while <Boolean Expression>:
    <do something>
```

## For each element in [1,2,3], print it.
```python
for item in [1,2,3]:
    print item
```

## While loop that does the same thing.
```python
item = 3
while item >= 1:
    print item
    item = item - 1
```

<table>
<thead>
<tr>
<th>Memory</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Name</td>
<td>Value</td>
<td></td>
</tr>
<tr>
<td>item</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
While Loops

**Def:** A *while* loop keeps iterating until a condition is not met.

```python
while <Boolean Expression>
    <do something>
```

## For each element in [1,2,3], print it.
```python
for item in [1,2,3]:
    print item
```

## While loop that does the same thing.
```python
item = 3
while item >= 1:
    print item
    item = item - 1
```

### Warning: you must ensure that your loop terminates!

*Infinite loops are no good.*

**Memory**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>item</td>
<td>0</td>
</tr>
</tbody>
</table>
Python Dictionaries

Task 1: Print the number of occurrences of the word That

myStr = 'That Sam I am That Sam I am I do not like That Sam I am'
myList = myStr.split()
numThats = 0
for word in myList:
    if word == 'That':
        numThats = numThats + 1
print numThats

Task 2: Print the number of occurrences of all words

<table>
<thead>
<tr>
<th>Word</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That</td>
<td>3</td>
</tr>
<tr>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
</tr>
<tr>
<td>am</td>
<td>3</td>
</tr>
<tr>
<td>do</td>
<td>1</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
</tr>
<tr>
<td>like</td>
<td>1</td>
</tr>
</tbody>
</table>
Python Dictionaries

Task 2: Print the number of occurrences of all words

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'That'</td>
<td>3</td>
</tr>
<tr>
<td>'Sam'</td>
<td>3</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That</td>
<td>3</td>
</tr>
<tr>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
</tr>
<tr>
<td>am</td>
<td>3</td>
</tr>
<tr>
<td>do</td>
<td>1</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
</tr>
<tr>
<td>like</td>
<td>1</td>
</tr>
</tbody>
</table>
# Python Dictionaries

Keys can be:
- Strings
- Integers
- Floats

Values can be:
- Integers
- Floats
- Strings
- Lists

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘That’</td>
<td>3</td>
</tr>
<tr>
<td>‘Sam’</td>
<td>3.0</td>
</tr>
<tr>
<td><code>‘Jack’</code></td>
<td>‘503-111-11111’</td>
</tr>
<tr>
<td>‘AUG’</td>
<td>‘M’</td>
</tr>
<tr>
<td>500</td>
<td>‘five hundred’</td>
</tr>
<tr>
<td>102</td>
<td>6</td>
</tr>
</tbody>
</table>
**Python Dictionaries**

A **dictionary** stores key-value pairs

```python
def: freq = {}
freq[‘That’] = 3
print freq
freq[‘Sam’] = 3
print freq
freq[‘like’] = 1
print freq

print freq[‘like’]
print freq[‘Sam’]

freq[‘That’] = freq[‘That’] + 1
print freq

print freq.keys()
print freq.values()
```

<table>
<thead>
<tr>
<th>Word</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That</td>
<td>3</td>
</tr>
<tr>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
</tr>
<tr>
<td>am</td>
<td>3</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
</tr>
<tr>
<td>like</td>
<td>1</td>
</tr>
</tbody>
</table>
Computing Word Frequencies

**Def:** A dictionary stores key-value pairs

```python
myStr = 'That Sam I am That Sam I am I do not like That Sam I am'
myList = myStr.split()
freq = {}
for word in myList:
    if word not in freq.keys():
        # write something here
    else:
        # write something here
print freq
```

Word Frequencies:

<table>
<thead>
<tr>
<th>Word</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>That</td>
<td>3</td>
</tr>
<tr>
<td>Sam</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
</tr>
<tr>
<td>am</td>
<td>3</td>
</tr>
<tr>
<td>do</td>
<td>1</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
</tr>
<tr>
<td>like</td>
<td>1</td>
</tr>
</tbody>
</table>
Computing Word Frequencies

Def: A dictionary stores key-value pairs

myStr = 'That Sam I am That Sam I am I do not like That Sam I am'
myList = myStr.split()
freq = {}
for word in myList:
    if word not in freq.keys():
        # write something here
    else:
        # write something here
print freq

<table>
<thead>
<tr>
<th>Word</th>
<th>Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>am</td>
<td>3</td>
</tr>
<tr>
<td>do</td>
<td>1</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
</tr>
<tr>
<td>like</td>
<td>1</td>
</tr>
</tbody>
</table>

Boolean Expression: Is the variable word in the list of keys from the dictionary freq?

Warning! The order of freq.keys() may change. Unlike lists, dictionaries don’t have a guaranteed order.