End Chapter 3

Skip Chapters 4 & 5

Start Chapter 6
What are Bits?
How is the Memory Organized?
(Part of Chapter 3 in Zelle)
Number Systems

0
1
2
3
4
5
6
7
8
9
10

MATH
## Number Systems

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
</tbody>
</table>
Number Systems

0   0
1   1
2   10
3   11
4   100
5   101
6   110
7   111
8   1000
9   1001
10  1010

MATH

BOOLEAN

LOGIC

Claude Shannon

True
False
AND
OR
NOT
Number Systems

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
</tr>
</tbody>
</table>

Claude Shannon

Boolean Logic

True
False
AND
OR
NOT

Math

Engineering
Number Systems
0  0
1  1
2  10
3  11
4  100
5  101
6  110
7  111
8  1000
9  1001
10 1010

Claude Shannon

MATH

BOOLEAN
LOGIC

ENGINEERING

True
False
AND
OR
NOT
Number Systems
0 0
1 1
2 10
3 11
4 100
5 101
6 110
7 111
8 1000
9 1001
10 1010

Claude Shannon

Boolean Logic
True False
AND OR
NOT

Math

Computers

Engineering

D. Thiebaut, Computer Science, Smith College
True -> AND -> False
False -> OR -> NOT
True → AND → True
True → OR → True
True → NOT → True
AND

OR

NOT

True

False

False

True
Can be built with a few transistors

Can use electricity to represent 0 and 1
1 → AND → 1 → NOT → 0
1 → NOT → 0 → AND → 0
1 → NOT → 0 → AND → 1
\[ \neg 0 \]

Diagram:

- Input: 0
- Output: 0
- Through AND: 1\rightarrow 1\rightarrow 0\rightarrow 0\rightarrow 1
- Through NOT: 1\rightarrow 0\rightarrow 0\rightarrow 0\rightarrow 1

Result: 0
A diagram showing a logical circuit with the following labels:

- The circuit starts with an input of 1.
- An AND gate takes an input of 0 and an output of 0, resulting in an output of 0.
- The output of 0 from the AND gate is input to a NOT gate, which inverts the 0 to 1.
- The output of 1 from the NOT gate is input to another AND gate, which takes an input of 1 and an output of 1, resulting in an output of 1.
- The output of 1 from the AND gate is connected back to the input of the NOT gate, creating a feedback loop.

The diagram also has a red number 1 outside the circuit, indicating the overall output of the circuit.
• A **bit** is a device that stores either 1 or 0
• A bit is a device that stores either 1 or 0
• By extension, a bit is either 1 or 0
• A bit is a device that stores either 1 or 0
• By extension, a bit is either 1 or 0
• A bit is a unit of information
• A bit is a device that stores either 1 or 0
• By extension, a bit is either 1 or 0
• A bit is a unit of information
• 2 bits take on 1 of 4 states: 00, 01, 10, 11
• A bit is a device that stores either 1 or 0
• By extension, a bit is either 1 or 0
• A bit is a unit of information
• 2 bits take on 1 of 4 states: 00, 01, 10, 11
• 3 bits: 000, 001, 010, 011, 100, 101, 110, 111
• A bit is a device that stores either 1 or 0

• By extension, a bit is either 1 or 0

• A bit is a unit of information

• 2 bits take on 1 of 4 states: 00, 01, 10, 11

• 3 bits: 000, 001, 010, 011, 100, 101, 110, 111

• 8 bits = 1 byte
  00000000, 00000001, … to 11111111
  256 possible combinations of 0s and 1s
End Chapter 3
Skip Chapters 4 & 5
Start Chapter 6
Chapter 6 in Zelle
Indexing in Strings

Indexing in Lists

Splitting Strings into Lists

String Methods
We like to keep information in numbered boxes in memory.

Data is kept in collections of “things”
For example: strings
Strings are collections of characters

name = "ALIBABA"
name = "ALIBABA"

Important conceptual change in the way we look at string
name

0 1 2 3 4 5 6

ALIBABA
name

A L I B A B A A

name[-6] = 'L'
A diagram illustrating the indexing of a string named "name". The string starts with 'A' and ends with 'A'. The indexing points to the characters 'L' and 'B' at positions -6 and -2 respectively.
There are two different ways to access the last character of a string. Which are they?
Demo Time!

```python
20
>>> c
30
>>> trio = a, b, c
>>> trio
(10, 20, 30)
>>> x, y, z = trio
>>> x
10
>>> y
20
>>> z
30
>>> i, j = trio
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    i, j = trio
ValueError: too many values to unpack
```
Logistic (lab cancelled)

Indexing in Strings

**Indexing in Lists**

Splitting Strings into Lists

String Methods
Strings are collections of characters
Lists are collections of various data types

animals = [ "pig", "hen", "dog", "cat" ]
animals = [ "pig", "hen", "dog", "cat" ]
animals = [ "pig", "hen", "dog", "cat" ]

animals[0]
animals = [ "pig", "hen", "dog", "cat" ]
animals[0]
animals = [ "pig", "hen", "dog", "cat" ]

animals[0]
animals[3]
animals = [ "pig", "hen", "dog", "cat" ]

animals[0]

animals[3]
animals = [ "pig", "hen", "dog", "cat" ]

animals[0]

animals[3]

animals[-3]
animals = ['pig', 'hen', 'dog', 'cat']

animals[0]

animals[3]

animals[-3]
Playing with Python Semantic

farm = ["pig", "dog", "horse", "hen"]

Find as many different ways to print all the animals in the farm as you can...
Slicing a String

name

A L I B A B A
Slicing a String

name = ALIBABA

section = name[1:4]
Slicing a String

name

section = name[1 : 4]

section

D. Thiebaut, Computer Science, Smith College
name

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
A & L & I & B & A & B & A \\
\end{array}
\]

name[0:1]  -->
name

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
A & L & I & B & A & B & A \\
\end{array}
\]

name[0:1]    —>    A
name

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
A & L & I & B & A & B & A
\end{array}
\]

name[0:1]  \rightarrow  A

name[5:6]  \rightarrow
The diagram illustrates a string named `name` with characters from `A` to `A`. It shows:

- `name[0:1]` maps to `A`.
### Name Array and Indexing

Consider the name array `ALIBABA

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>A</td>
<td>L</td>
<td>I</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

#### Indexing Examples

- `name[0:1]` → A
- `name[5:6]` → B
- `name[-2:-1]` → A

This demonstrates how to access substrings using slicing in Python, where `name[0:1]` refers to the first 1 character starting at index 0, `name[5:6]` refers to the 5th and 6th characters, and `name[-2:-1]` refers to the last 2 characters starting from the end of the array.
name

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
A & L & I & B & A & B & A \\
\end{array}
\]

-7 -6 -5 -4 -3 -2 -1

name[0:1]  \rightarrow  A

name[5:6]  \rightarrow  B

name[-2:-1]  \rightarrow  B

name[0:-1]  \rightarrow  }
name: ALIBABA

name[0:1]  -->  A

name[5:6]  -->  B

name[-2:-1]  -->  B

name[0:-1]  -->  ALIBABA
Two Special Slices

\[
\text{part} = \text{name}[5]
\]

\[
\text{part} = \text{name}[3:]
\]
Two Special Slices

\[
part = \text{name}[0:5] \quad \leftarrow \text{from beginning to 5}
\]

\[
part = \text{name}[3:] \quad \leftarrow \text{from 3 to end, including last}
\]
name = ALIBABA

name[ :4]  -->
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>L</td>
<td>I</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

name[ :4] → ALIB
name

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\end{array}
\]

name[ :4]  --->

\[
\begin{array}{cccc}
A & L & I & B \\
\end{array}
\]

name[ 3: ]  --->

D. Thiebaut, Computer Science, Smith College
name: ALIBABA

name[0:4] -> ALIB

name[3:] -> BABA
Interesting Property

name = "Some string of characters"

name2 = name[ : 7] + name[ 7 : : ]
Interesting Property

name = “Some string of characters”

name2 = name[ : n] + name[ n : ]

name2 contains the same string as name
Extract the **drive** and **extension** information from a file name:

name = “H:/Documents/solutionsHw4.doc”

Replace the “doc” extension by “txt” in the file name:

name = “H:/Documents/solutionsHw4.doc”

Get the first and last name of a person and create a computer account with the first letter of the first name, and the last name, concatenated.
Transforming dates:
Transform a string, such as “02162015” into 16 Feb 2015.

Exercises

Get the first and last name from a person, and display a “triangle” made of her full name. For example,
fname = “Maria”
lname = “LUCE”
Output of program:
M
Ma
Mar
Mari
Maria
MariaL
MariaLU
MariaLUC
MariaLUCE
Good place to stop!...
Review
• Strings are lists of characters
• Strings are lists of characters

name  A L I B A B A
• Strings are **lists** of characters

```plaintext
name
```

A L I B A B A

• Lists are *lists* of items, too!
• Strings are **lists** of characters

  name  
  A  L  I  B  A  B  A

• Lists are *lists* of items, too!

  farm = [ "dog", "cat", "pig" ]
• Strings are **lists** of characters

```
name  A L I B A B A
```

• Lists are *lists* of items, too!

```
farm = [ "dog", "cat", "pig" ]
```

• They can be indexed, and sliced
• Strings are **lists** of characters

```
name A L I B A B A
```

• Lists are *lists* of items, too!

```
farm = [ "dog", "cat", "pig" ]
```

• They can be indexed, and sliced

```
name[-1]    name[0:2]
farm[-1]    farm[0:2]
```
• Strings are **lists** of characters

```
name = A L I B A B A
```

• Lists are *lists* of items, too!

```
farm = [ "dog", "cat", "pig" ]
```

• They can be indexed, and sliced

```
name[-1]  name[0:2]  A L

"pig"  farm[-1]  farm[0:2]  [ "dog", "cat"]
```
Lists and Strings behave similarly but are different in an important way.
>>> farm = ["dog", "cat", "pig"]

>>> farm[ -1 ]
'pig'

>>> farm[ 2:3 ]
['pig']

>>> farm[ 1:3 ]
['cat', 'pig']

>>> farm[ 1 ] = "hen"

>>> farm
['dog', 'hen', 'pig']
Python 3.1.1 (r311:74543, Aug 24 2009, 18:44:04)
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for more information.

>>> farm = [ "dog", "cat", "pig" ]

>>> farm[ -1 ]
'pig'

>>> farm[ 2:3 ]
['pig']

>>> farm[ 1:3 ]
['cat', 'pig']

>>> farm[ 1 ] = "hen"

>>> farm
['dog', 'hen', 'pig']

>>> name = "Alibaba"

>>> name[ -1 ]
'a'

>>> name[ 0 ]
'A'

>>> name[ -3:-1 ]
'ab'

>>> name[ 3 ] = 'Z'
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    name[3] = 'Z'
TypeError: 'str' object does not support item assignment

>>> |
We cannot modify a string!
Strings are Immutable
Lists (with [...]) are mutable
Lists (with (…)) are immutable
Logistic (lab cancelled)

Indexing in Strings

Indexing in Lists

**String Objects and Methods**

Splitting Strings into Lists
Objects
Objects

data
Objects

action

data
Examples
Examples

"{0:1}@{1:1}"
Examples

format action
("max", "gmail.com")
Examples

format action
("max", "gmail.com")

max@gmail.com
Examples

`"{0:1}@{1:1}".format( "max", "gmail.com" )`
Examples

```
"{0:1}@{1:1}".format("max", "gmail.com")
```

format action
("max", "gmail.com")
Logistic (lab cancelled)

Indexing in Strings

Indexing in Lists

String Objects and Methods

Splitting Strings into Lists
• `upper()`

"hello there"
• `upper()`
• upper()
• `upper()`

```
"hello there".upper()
```

`upper()`
• upper()

• lower()

• center( n )

• capitalize()

• title()
- upper()
- lower()
- center(n)
- capitalize()
- title()

- format( ..., ...)
- find( ...)
- replace( ..., ...)

D. Thiebaut, Computer Science, Smith College
Keyword = "Built-In Types"

https://docs.python.org/3.1/library/stdtypes.html?highlight=capitalize#string-methods
>>> a = """

DEMO TIME

"""
>>> |

Ln: 17 Col: 4
Multiple Transformations
"hello there".upper()

upper()

"HELLO THERE"
"hello there".upper()
"hello there".upper()
"hello there".upper().center(20)
Write a program that **prompts** the user for her first and last name, and **prints** both, in the **proper case**, **centered** in 60 spaces.

**Exercises**

(Part 1)

Write a program that takes a string, where a **phone number** is located. The phone number always **start at Index 7**, and contains 10 numbers (no spaces). **Print** the **phone number only**, in the form: **(xxx) xxx-xxxx**.
Write a program that takes the string book (above), and prints it, the title centered, all caps, in a line of 60 chars, and the author’s name, capitalized, and centered in 60 chars, followed by a blank line, followed by the first sentence (whichever way it comes out).

book = """"Ulysses
James Joyce
Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed.
"

Project Gutenberg
http://www.gutenberg.org/cache/epub/4300/pg4300.txt
We stopped here last time...
Split(), the workhorse of string methods

Functions

Function Parameters

Functions Returning Values
split()
line = "The quick red fox jumped over the dog"
line = "The quick red fox jumped over the dog"

line.split( '' )
line = "The quick red fox jumped over the dog"

line.split( ' ' )
line = "The quick red fox jumped over the dog"

line.split( ' ' )

["The", "quick", "red", "fox", "jumped", "over", "the", "dog" ]
```python
line = "The quick red fox jumped over the dog"

line.split( 'o' )

["The quick red f",
 "x jumped",
 "ver the d",
 "g" ]
```
line = "hello Roth"

line.split( 'h' )

["", "ello Rot", "]
```python
line = """The quick red fox jumped over the lazy brown sleeping dog"""

line.split()  

[ "The quick red fox", "jumped over", "the lazy brown sleeping dog" ]
```
line = """The quick red fox
jumped over
the lazy brown sleeping dog"""

line.split( "\n" )

[ "The quick red fox",
"jumped over",
"the lazy brown sleeping dog" ]
line = """The quick red fox
jumped over
the lazy brown sleeping dog"""

line = "The quick red fox
jumped over
the lazy...dog"

Equivalent

(no room on slide!)
Exercise

```python
poem = """"Chocolate
Chocolate is the first luxury.
It has so many things wrapped up in it:
Deliciousness in the moment,
childhood memories,
and that grin-inducing
feeling of getting a reward for being good.
--Mariska Hargitay"
"

# display each line centered in 60 spaces.
# first line all uppercase.
# last line right justified in 60 spaces.
```
Split(), the workhorse of string methods

Functions

Function Parameters

Functions Returning Values
Functions

are your minions!
They have names…
(def main)
They work only if you call them...  
main()
They can work on different things…

daveWorkOn( "Banana" )
daveWorkOn( "Lots of Bananas" )
A Real Example

http://cs.smith.edu/dftwiki/index.php/MakeStaticPackage.py_Source_Code
main()

functions
dave() carl() stuart()
Two different types of functions:
Some functions always do the same thing

def printBar():
    print( 60 * '-' )

def sayHello():
    print( )
    print( "Hello, and welcome!" )
    print( )
Some functions always do the same thing

```python
def printBar():
    print( 60 * '-' )

def sayHello():
    print( )
    print( "Hello, and welcome!" )
    print( )
```
Some functions adjust their behavior depending on what we give them to work with.
work for Dave to perform:

**Eat**( fruit )
- open mouth
- put *fruit* in mouth
- chew
- swallow
work for Dave to perform:

**Eat** (fruit )
- open mouth
- put *fruit* in mouth
- chew
- swallow

Dave, eat( banana )
work for Dave to perform:

**Eat( fruit )**
- open mouth
- put *fruit* in mouth
- chew
- swallow

Dave, eat( banana )

Dave, eat( orange )
work for Dave to perform:

Eat( fruit )
- open mouth
- put fruit in mouth
- chew
- swallow

Dave, eat( banana )

Dave, eat( orange )

Dave, eat( apple )
def daveEats( fruit):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )

def main():
    daveEats( "banana" )

main()

Dave opens his mouth and eats the banana
def daveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )

def main():
    daveEats( "banana" )
    daveEats( "apple" )
main()

Dave opens his mouth and eats the banana
Dave opens his mouth and eats the apple
def daveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )

def main():
    daveEats( "banana" )
    daveEats( "apple" )

main()
def daveEats( fruit ):
    print( "Dave opens his mouth" )
    print( "and eats the", fruit )

def main():
    daveEats( "banana" )
    daveEats( "apple" )

main()
def daveEats( fruit ):  
    print( "Dave opens his mouth" )  
    print( "and eats the", fruit )

def main():  
    daveEats( "banana" )  
    daveEats( "apple" )

main()
Another Example

def printBar( char, length ):  
    print( char * length )

def main():  
    printBar( "#", 10 )

main()
Another Example

def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )

main()
Another Example

def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )
    printBar( "a", 5 )

main()

##########
Another Example

def printBar(char, length):
    print(char * length)

def main():
    printBar("#", 10)
    printBar("a", 5)

main()
Exercise

def printBar( char, length ):
    print( char * length )

def main():
    printBar( "#", 10 )
    printBar( "a", 5 )
    printBar( 3, 5 )
main()
Write a program that sings happy birthday to **Dave**.

Write a program that sings happy birthday to **Dave** and to **Stuart**.

Let’s sing

Happy Birthday to some minions
Modify the program so that it sings happy birthday to Dave, Stuart, Jerry, Jorge, Tim, Mark, Phil, Kevin, and Jon.

*For you to play:* Modify the program so that it sings happy birthday, and formats the song with bars made with the person’s name, e.g. *Dave*Dave*Dave*Da… above and below the message.
We stopped here last time…