Chapter 3
in Zelle
Finish Exercise from last week (printing bar graph)…
Midterm Exam: Friday Oct. 16th

In class, 1 hour 10 min

On paper

Closed books

Closed notes

Closed computers
When computers were human, a talk by David Grier: start at 28m12s, for 4 minutes.  
http://youtu.be/YwqltwvPnkw?t=28m12s
Arithmetic operators and math functions

Printing numbers to look "nice"

Using a main() function

Accumulating results

What are bits?
# Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>sum</td>
<td>float if 1 side is float, else int</td>
</tr>
<tr>
<td>-</td>
<td>subtraction</td>
<td>float if 1 side is float, else int</td>
</tr>
<tr>
<td>*</td>
<td>multiplication</td>
<td>float if 1 side is float, else int</td>
</tr>
<tr>
<td>/</td>
<td>real division</td>
<td>float</td>
</tr>
<tr>
<td>//</td>
<td>integer division</td>
<td>int</td>
</tr>
<tr>
<td>%</td>
<td>modulo</td>
<td>int</td>
</tr>
<tr>
<td>**</td>
<td>exponentiation</td>
<td>float if 1 side is float, else int</td>
</tr>
</tbody>
</table>
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
```
Exercise:

Writing a Teller Machine Program
How much money do you want to withdraw?
How much money do you want to withdraw? 139
How much money do you want to withdraw? 139

Please find the following bills below:
6 $20-bill(s)
1 $10-bill(s)
1 $5-bill(s)
4 $1-bill(s)
Programming 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
```
Arithmetic operators and math functions

Printing numbers to look "nice"

Using a main() function

Accumulating results

What are bits?
name: Alex age: 12 coefficient: 6.18888888 /20
name: Anastasia age: 22 coefficient: 8 /20
Current output

name: Alex age: 12 coefficient: 6.18888888/20
name: Anastasia age: 22 coefficient: 8/20
(See Section 5.8.2 in Zelle)
name = "Alex"
"name: {0:10}X" . format( name )
name = "Alex"
"name: {0:10}X" . format( name )

'name: X'
1234567890
name = "Alex"
"name: {0:10}X". format( name )

'name: X'
1234567890

{name: Alex X'
1234567890
name = "Alex"
"name: {0:10}X" . format( name )

'name: X'
1234567890

'name: Alex X'
123456789
name1 = "Alex"
name2 = "Anastasia"
"names: {0:10}X{1:10}Y" . format( name1, name2 )
name1 = "Alex"
name2 = "Anastasia"
"names: {0:10}X{1:10}Y" . format( name1, name2 )

"names: X Y"
1234567890 1234567890
name1 = "Alex"
name2 = "Anastasia"
"names: {0:10}X{1:10}Y" . format( name1, name2 )

"names: X Y"
1234567890 1234567890

"names: Alex XAnastasia Y"
1234567890 1234567890
name1 = "Alex"
name2 = "Anastasia"

"names: {0:10}X{1:10}Y". format( name1, name2 )

"names: X         Y"
1234567890 1234567890

"names: Alex      XAnastasia Y"
1234567890 1234567890
name1 = "Alex"
name2 = "Anastasia"
"names: {0:10}X{1:10}Y" . format( name1, name2 )

"names: X Y"
1234567890 1234567890

"names: Alex XAnastasia Y"
1234567890 1234567890

What about right-justification?
We stopped here last time...
name1 = "Alex"
name2 = "Anastasia"

"names: {0:>10}X{1:>10}Y". format( name1, name2 )

"names:       X        Y"
1234567890  1234567890

"names:     AlexX  AnastasiaY"
1234567890  1234567890

"names:     AlexX  AnastasiaY"
Bar Graph Exercise (revisited)

First name? Dominique
Last name? Thiebaut
Id? 990123456
Final grade? 90

```
+-----------------------------------------------------+
| Dominique Thiebaut                                  |
| 990123456                                           |
+-----------------------------------------------------+

00...10...20...30...40...50...60...70...80...90...100

grade: ################################################
class: ################################################
It works the same for integers!
name1 = "Alex"
age = 22
"name: {0:10} age: {1:3}!". format( name1, age )

"name:     age:   !
1234567890  123

"name: Alex age: 22!"
1234567890  123

"name: Alex age: 22!"

Ints are automatically right-aligned
name1 = "Alex"
age = 22
"name: {0:10} age: {1:<3}!". format( name1, age )

"name: Alex       age: 22 !"

Use '<' to left-align
Floats are a bit different… We need to specify

- a total number of digits, and
- a number of digits after the decimal point.
\[
\pi = 3.141592653589793
\]

"\pi ={0:10.2f}#" . format( \pi )

\[
\pi = 1234567890
\]
\[ \pi = 3.141592653589793 \]

"\pi ={0:10.2f}\#" . \texttt{format}( \pi )

"\pi =\#"
\begin{verbatim}
 1234567890
  12
\end{verbatim}

"\pi = 3.14\#"
\begin{verbatim}
 1234567890
  12
\end{verbatim}
We can left- and right-align floats with '<' and '>' as well...
name1 = "Alex"
age1 = 12
coef1 = 6.188888
name2 = "Anastasia"
age2 = 22
coef2 = 8

print( "name: {0:10} age: {1:2} coefficient: {2:8.2f}"
      .format( name1, age1, coef1 ) )
print( "name: {0:10} age: {1:2} coefficient: {2:8.2f}"
      .format( name2, age2, coef2 ) )

name: Alex age: 12 coefficient: 6.19/20
name: Anastasia age: 22 coefficient: 8.00/20
Arithmetic operators and math functions

Printing numbers to look "nice"

Using a `main()` function

Accumulating results

What are bits?
```python
print( "Hello world!" )
print( "Welcome to CSC111!" )

def main():
    print( "Hello world!" )
    print( "Welcome to CSC111" )

main()
```
def main():
    print("Hello world!")
    print("Welcome to CSC111")

main()
```python
def main():
    print("Hello world!")
    print("Welcome to CSC111")
main()
main()
```

Hello world!
Welcome to CSC111
Hello world!
Welcome to CSC111
Function Syntax

def <name> ( <parameters> ):
<body>
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
```
Arithmetic operators and math functions

Printing numbers to look "nice"

Using a `main()` function

**Accumulating results**

What are bits?
Compute the Sum of Several Numbers.

\[
\text{ages} = [12, 14, 10] \\
\text{sumAge} = ?
\]
Compute the Sum of Several Numbers.

```python
ages = [12, 14, 10]

sumAge = ?

# review printing all elements of a list
for age in [12, 14, 10]:
    print(age)
```

```
12
14
10
```
Compute the Sum of Several Numbers.

```python
ages = [12, 14, 10]
sumAge = ?

# review printing all elements of a list
for age in [12, 14, 10]:
    print(age)
```

```
0  ← sumAge
12
0+12 = 12  ← sumAge
14
12+14 = 26  ← sumAge
10
26+10 = 36  ← sumAge
```

← Algorithm for sum
Compute the Sum of Several Numbers.

```python
ages = [12, 14, 10]
sumAge = 0

# review printing all elements of a list
for age in [12, 14, 10]:
    # print(age)
    sumAge = sumAge + age

0                  <-- sumAge
0+12 = 12           <-- sumAge
12+14 = 26          <-- sumAge
26+10 = 36          <-- sumAge
```
Compute the Sum of Several Numbers.

```python
ages = [12, 14, 10]
sumAge = 0

# review printing all elements of a list
for age in [12, 14, 10]:
    # print( age )
    sumAge = sumAge + age

print( "sum = ", sumAge )
```

sum =  36
Exercises

Compute is the sum of all the integers between 1 and 10

Compute is the sum of all the integers between 1 and 1000

Compute is the sum of all the multiples of 6 between 0 and 100, included
More Exercises

Using the same approach, write a loop that creates a string of special characters, defined by a list of values:
Example:
  [ 3, 2, 1, 5] would result in "###+++#####+++"

  [1,2,1,3,4] would result in "###++####+++++++"
Arithmetic operators and math functions

Printing numbers to look "nice"

Using a `main()` function

Accumulating results

**What are bits?**
Number Systems

0
1
2
3
4
5
6
7
8
9
10

MATH
<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>
<tr>
<td>Number Systems</td>
<td>00</td>
</tr>
<tr>
<td>----------------</td>
<td>----</td>
</tr>
<tr>
<td>True</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td></td>
</tr>
<tr>
<td>AND</td>
<td></td>
</tr>
<tr>
<td>OR</td>
<td></td>
</tr>
<tr>
<td>NOT</td>
<td></td>
</tr>
</tbody>
</table>

**Math** overlaps with **Boolean Logic**.
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
### Number Systems

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
</tr>
<tr>
<td>1</td>
<td>01</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**

- True
- False

**LOGIC**

- 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

D. Thiebaut, Computer Science, Smith College
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

True False
AND OR
NOT

MATH

BOOLEAN
LOGIC

COMPUTERS

ENGINEERING
True → AND → True

True → OR →

True → NOT →
Can be built with a few transistors

Can use electricity to represent 0 and 1
\( X = 0 \)

\[
\begin{array}{ccc}
& & 1 \\
\text{AND} & \rightarrow & \text{NOT} \\
1 & \rightarrow & 0 \\
\text{NOT} & \leftarrow & 0 \\
1 & \leftarrow & \text{AND} \\
& & 1
\end{array}
\]
\[ 0 \]

Diagram showing a circuit with AND and NOT gates.
• 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
https://www.youtube.com/watch?v=xatYwFdzR0k
Character: A

Pixel:

<table>
<thead>
<tr>
<th>RED</th>
<th>GREEN</th>
<th>BLUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10001000</td>
<td>01101010</td>
<td>00001000</td>
</tr>
</tbody>
</table>