Lecture Notes
CSC111

Week 3 — Sept 21, 2015

Dominique Thiébaut
dthiebaut@smith.edu
Chapter 3
in Zelle
Programming Tips

- Never try to solve the whole problem at once
- Figure out how to solve smaller problems and merge pieces of code together
- Replace inputs by assignments until the last steps
- Make the program print intermediate values as debugging help. Remove these print statements at the end.
WHAT ARE THE 10 MOST FAMOUS SOFTWARE PROGRAMS WRITTEN IN PYTHON?

by HSG on Mar 19, 2014 in Articles from Software Fans

Python is an incredibly powerful and useful computer programming language that many of the biggest websites in the world rely on for their foundation. Python provides reliable results that are functional and involve a variety of dynamic scripted and non-scripted contexts. And because it is free and open source, it has remained a popular choice for a variety of different developers who are looking to build new sites on one of the most reliable languages available. Here is a look at 10 of the most famous software programs that are written in Python and what they do.

YouTube
Dropbox
Google
Quora
Instagram
BitTorrent
Spotify
Reddit
Yahoo Maps
Hipmunk

http://www.hartmannsoftware.com/Blog/Articles_from_Software_Fans/Most-Famous-Software-Programs-Written-in-Python
Information about Midterm Exam:

Timed, on Moodle

Closed books

Closed notes

Closed Web

Closed Idle
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 Shell

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
>>> coef = 6.1888888

>>> print("name:", name, "age:", age, "coefficient:", coef, "/20")
name: Alex age: 12 coefficient: 6.1888888 /20

>>>
name: Alex age: 12 coefficient: 6.18888888 /20
name: Anastasia age: 22 coefficient: 8 /20
name: Alex age: 12 coefficient: 6.18888888 /20
name: Anastasia age: 22 coefficient: 8 /20
name: Alex age: 12 coefficient: 6.18888888 /20
name: Anastasia age: 22 coefficient: 8 /20

name: Alex age: 12 coefficient: 6.19/20
name: Anastasia age: 22 coefficient: 8.00/20

(See Section 5.8.2 in Zelle)
name = "Alex"
"name: {0:10}X" . format( name )
• The dot . is an important operator
• It works with objects
• In Python, strings are objects

"hello {0:10} there!" . format( name )

--> "String, please format yourself, and substitute the contents of name inside your brackets"
name = "Alex"
"name: {0:10}X"  . format( name )
name = "Alex"
"name: {0:10}X" . format( name )

'label:
 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

'name: Alex X'
We stopped here last time...
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 )

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

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

"names: X Y"
1234567890 1234567890

"names: Alex X Anastasia Y"
1234567890 1234567890

What about right-justification?
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"
<table>
<thead>
<tr>
<th>Dominique Thiebaut</th>
<th>990123456</th>
</tr>
</thead>
</table>

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

Ints are automatically right-aligned
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 !"
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}#" . format( Pi )

"Pi =      3.14#"
1234567890
   12

"Pi = 3.14#"
1234567890
  12
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( ??? .format( name1, age1, coef1 ) )
print( ??? .format( name2, age2, coef2 ) )

ame:   Alex age:  12 coefficient:   6.19/20
name: Anastasia age:  22 coefficient:   8.00/20
1234567890      123              123456
12
Applications

• Printing temperatures

• Be flexible!

• Temporary variables are good!

• Print a conversion table of temperatures using the {...} formatting command.
Exercises

Figure out the Right Print Format
## Exercise 1

```python
for i in range(10):
    print(i, 2**i)
```

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>32</td>
</tr>
<tr>
<td>6</td>
<td>64</td>
</tr>
<tr>
<td>7</td>
<td>128</td>
</tr>
<tr>
<td>8</td>
<td>256</td>
</tr>
<tr>
<td>9</td>
<td>512</td>
</tr>
</tbody>
</table>
Exercise 2

```python
count = 1
    print(count, name, len(name))
count = count + 1
```

```python
count = 1
    print(
        len(Doc ) = 3
        len(Grumpy ) = 6
        len(Happy ) = 5
        len(Sleepy ) = 6
        len(Dopey ) = 5
        len(Bashful)= 7
        len(Sneezy ) = 6
    )
count = count + 1
```
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>
Demonstration

```python
>>> 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.

ages = [12, 14, 10]
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
```
We stopped here last time...
狗年大吉！

代码写完了吗
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 5 between 0 and 100, included
Exercise

Compute is the average of all the integers in [100, 95, 100, 90, 60, 85]

Compute the total length of all the words in ["Doc", "Grumpy", "Happy", "Sleepy", "Dopey", "Bashful", "Sneezy"]
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 "#++#+++++++"

Compute is the factorial of 5
(factorial of 5 = 5 x 4 x 3 x 2 x 1)
Logistics

- Solution Programs appear after deadline at bottom of Homework page and Lab page
- Where are the TAs?
- Run & Evaluate
- No lab next week (Rally Day)
Arithmetic operators and math functions

Printing numbers to look "nice"

Using a \texttt{main()} function

Accumulating results

\textbf{What are bits?}
Number Systems

0
1
2
3
4
5
6
7
8
9
10

MATH
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>
Number Systems

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

BOOLEAN
LOGIC

Math

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

MATH

BOOLEAN LOGIC

Claude Shannon, 1937
1 2 3
+ 2 9 6
-------
1 2 3
+ 2 9 6

---------
9
\[ \begin{array}{c}
1 \\
1 2 3 \\
+ 2 9 6 \\
\hline
1 9
\end{array} \]
\[
\begin{array}{c}
1 \\
1 \ 2 \ 3 \\
+ \ 2 \ 9 \ 6 \\
\multicolumn{3}{c}{\text{———}} \\
4 \ 1 \ 9
\end{array}
\]
1
7
+  8
---------

7
+ 8
_______
\[
7 + 8
\]

\[
\text{———}
\]
1
7
+ 8
_______
5
\[
\begin{array}{c}
1 \\
1 \ 2 \ 3 \\
+ \ 2 \ 9 \ 6 \\
\hline \\
4 \ 1 \ 9
\end{array}
\]

\[
\begin{array}{c}
0 \ 1 \ 1 \\
0 \ 1 \ 1 \\
+ \ 1 \ 1 \ 0 \\
\hline \\
1 \ 0 \ 0
\end{array}
\]
\[
\begin{array}{c}
\text{1} \\
1 \quad 2 \quad 3 \\
+ \quad 2 \quad 9 \quad 6 \\
\hline
4 \quad 1 \quad 9
\end{array}
\]

\[
\begin{array}{c}
\text{0} \\
0 \quad 1 \quad 1 \\
+ \quad 1 \quad 1 \quad 0 \\
\hline
1 \quad 1 \quad 0 \quad 0
\end{array}
\]
\[
\begin{array}{c}
1 \\
1 2 3 \\
+ 2 9 6 \\
\hline
4 1 9 \\
\end{array}
\quad \quad \quad
\begin{array}{c}
1 \\
0 1 1 \\
+ 1 1 0 \\
\hline
0 1 \\
\end{array}
\]

0
1
10
11
100
<table>
<thead>
<tr>
<th>Number Systems</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>

**Boolean Logic**

- True
- False
- AND
- OR
- NOT

Claude Shannon, 1937
Number Systems

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

BOOLEAN

LOGIC

Claude Shannon

True
False
AND
OR
NOT

ENGINEERING

Electricity

ON
OFF
electronic
circuits for
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>
<tr>
<td>11</td>
<td>1011</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
</tr>
</tbody>
</table>

Boolean Logic

- True
- False
- AND
- OR
- NOT

Claude Shannon

Mathematics

- Electricity
- ON
- OFF
- electronic circuits for
- AND, OR, NOT

Engineering
Boolean Operators

| AND |  
|-----|---
| a   | b | a and b |

<table>
<thead>
<tr>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
</tbody>
</table>
Boolean Operators

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

- **AND**
- **OR**
- **NOT**
Boolean Operators

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

AND

OR

NOT
Boolean Operators

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>
Boolean Operators

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a and b</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a and b</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td>a and b</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>---------</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
\[
\begin{align*}
\text{AND} & \quad \text{OR} & \quad \text{NOT} \\
\text{False} & \quad \text{True} & \quad \text{True} \\
\text{True} & \quad \text{False} & \quad \text{False} \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a or b</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>
True ——> 1
False ——> 0

\[
\begin{array}{c}
\text{AND} \\
1 & 0 & \rightarrow & 0
\end{array}
\]

\[
\begin{array}{c}
\text{OR} \\
1 & 0 & \rightarrow & 1
\end{array}
\]

\[
\begin{array}{c}
\text{NOT} \\
0 & 0 & \rightarrow & 1
\end{array}
\]
True $\rightarrow 1$
False $\rightarrow 0$

<table>
<thead>
<tr>
<th>$a$</th>
<th>$b$</th>
<th>$a$ or $b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Can be built with a few transistors

Can use electricity to represent 0 and 1
\[ \begin{array}{c}
\text{AND} & \xrightarrow{0} & \text{NOT} & \xrightarrow{0} & \emptyset 1 \\
\text{NOT} & \xrightarrow{0} & \emptyset 1 & \xrightarrow{0} & \text{AND} & \xrightarrow{\emptyset 1} & 1
\end{array} \]
• 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
Character

Pixel

RED  GREEN  BLUE
10001000  01101010  00001000