Designing Functions

**What makes a good function?**

- Simplifies the code that calls it
- Isolates a design decision (easier to change)
- Used more than once
- Can be tested separately

... 

*A good function may have only some of these properties. Few have all.*

(Review from week 1)

def diff (a, b) :
    return a – b

... *then in main* ...

x = 5
y = 7

c = diff( x, y);

def diff (a, b) :
    return a – b

... *then in main* ...

x = 5
y = 7

c = diff( x, y);
a = 42
m = [12, 13, 14]

Environment:

\[
\begin{array}{c|c}
\text{a} & 42 \\
\hline
\text{m} & 12 \rightarrow 13 \rightarrow 14 \\
\end{array}
\]

\[
\begin{array}{c|c}
\text{a} & 42 \\
\hline
\text{m} & 12 \rightarrow 13 \rightarrow 14 \\
\text{n} & 12 \rightarrow 13 \rightarrow 14 \\
\text{k} & 12 \rightarrow 13 \rightarrow 14 \\
\end{array}
\]

Environment:

\[
\begin{array}{c|c}
\text{a} & 42 \\
\hline
\text{m} & 12 \rightarrow 13 \rightarrow 14 \\
\text{n} & 12 \rightarrow 13 \rightarrow 14 \\
\text{k} & 12 \rightarrow 12 \rightarrow 14 \\
\end{array}
\]

Environment:

\[
\begin{array}{c|c}
\text{a} & 42 \\
\hline
\text{m} & 12 \rightarrow 13 \rightarrow 14 \\
\text{n} & 12 \rightarrow 13 \rightarrow 14 \\
\text{k} & 12 \rightarrow 42 \rightarrow 14 \\
\end{array}
\]
Designing functions

A function can return a result
A function can modify an object

Rule of thumb:
Do one or the other ... don’t do both

def clobber(x):
    x[1] = 42
m = [12, 13, 14]
clobber(m)

Environment:
clobber:
x
main:
m

Any other ways for a function to affect other variables?

(year)
vowels = ['a', 'e', 'i', 'o', 'u']
def messup():
    vowels[1] = 'x'
    return
def main():
    print(vowels)
    messup()
    print(vowels)
main()
Slightly better:

```python
vowels = ['a', 'e', 'i', 'o', 'u']

def messup():
    global vowels
    vowels[1] = 'x'
    return

def main():
    print(vowels)
    messup()
    print(vowels)

main()
```

*Do this rarely and only when alternatives are uglier. Always document it.*

def bar(x):
    x = x * 3
    return x

def foo(x):
    y = bar(x)
    x = y + y
    return x

x = 2
y = foo(x)

(main)  x  2

(main)  x  2

(foo)  x  2

(foo)  x  2

(main)  x  2

(main)  x  2
```python
def bar(x):
    x = x * 3
    return x

def foo(x):
    y = bar(x)
    x = y + y
    return x

x = 2
y = foo(x)
```

The stack of activation records (frames):

1. **(main)** $x = 2$
2. **(foo)** $x = 2$
3. **(bar)** $x = 6$
def bar(x):
    x = x * 3  
    return x

def foo(x):
    y = bar(x)  
    x = y + y
    return x

x = 2
y = foo(x)

(main) x 2  

(foo) x 12

(y) 6

(main) x 2

(foo) x 2

(y) 6

(main) x 12

(foo) x 12

(y) 12
Do this yourself ...

Go to http://www.pythontutor.com/visualize.html
(seems to work best in Chrome)

Edit code, watch it execute

def bar(x):
    x = x * 3
    return x

def foo(x):
    y = bar(x)
    x = y + y
    return x

x = 2
y = foo(x)