Lists Revisited!

```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

Lists Revisited!

```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

Lists Revisited!

```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

Lists Revisited!

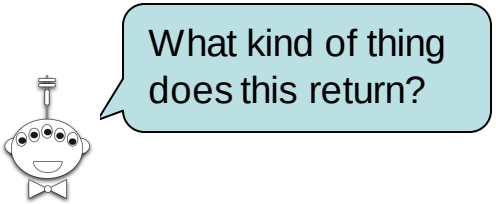
```
>>> L = [1, 42, 3, 4]
>>> L
[1, 42, 3, 4]
>>> L + 10
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: can only concatenate list (not "int") to list
>>> L + [50]
[1, 42, 3, 4, 50]
>>> L
[1, 42, 3, 4]
>>> L*2
[1, 42, 3, 4, 1, 42, 3, 4]
>>> M = [42, "hello", 3+2j, 3.141, [1, 2, 3, 4, 5, 6]]
```

L doesn't change!

Lists are "polymorphic"

List Indexing and Slicing!

```
            0   1    2    3
>>> M = [42, 3, 98, 37 ]

>>> M[0]

>>> M[2]

>>> M[0:2]

>>> M[0:3:2]

>>> M[1:]

>>> M[:-1]

>>> M[1:-2]

>>>
```



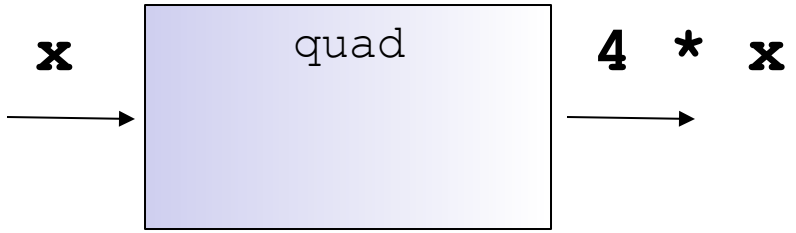
Try to reverse the list!

Strings Revisited

```
>>> S = "I love Spam!"
          0 1 2 3 4 5 6 7 8 9 1 1 1
                            0 1 2
>>> S[0]

>>> S[13]

>>> S[2:6]

>>> S[12:6:-1]
```

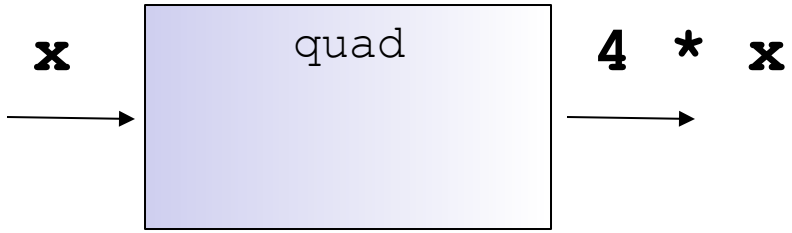
Composition of functions

```
def quad(x):
  return 4 * x
```

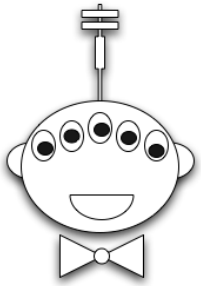


Composition of functions

```
def quad(x):
    return 4 * x
```

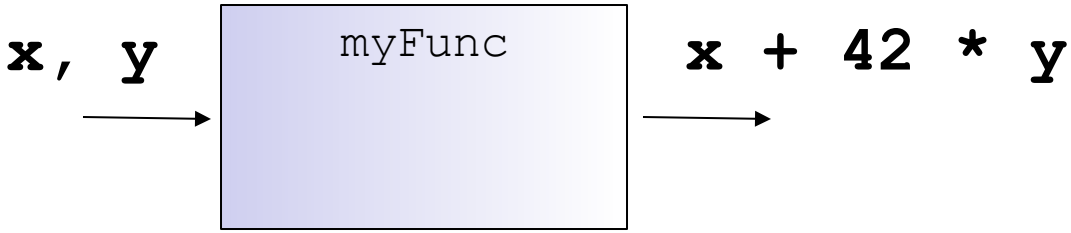


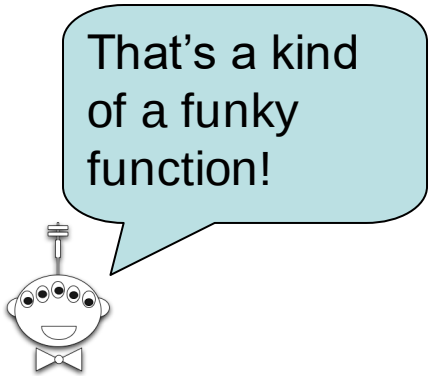

```
def quad(x):
    return dbl(dbl(x))
```



Doubly cool! (draw the boxes)

Multiple inputs...






```
# myFunc
# Authors: Ran Libeskind-Hadas
# Date:    August 27, 2012

def myFunc(x, y):
  """returns x + 42 * y"""
  return x + 42 * y
```

return **VS** print…

```
def dbl(x):
    return 2 * x
```

```
def trbl(x):
    print(2 * x)
```

```
def happy(input):
    y = dbl(input)
    return y + 42
```

```
def sad(input):
    y = trbl(input)
    return y + 42
```


```
def friendly(input):
    y = dbl(input)
    print(y, "is very nice!")
    return y + 42
```

Strings are in single or double quotes

Function Exercises

Exercise 1

Write a function named addTwoDigits that given a two-digit integer n, returns the sum of its digits. For example n = 29, the output should be addTwoDigits(n) = 11.

Exercise 2

Given an integer n, return the largest number that contains exactly n digits.For example n = 2, the output should be largestNumber(n) = 99.

Exercise 3

Write a function named reverse that takes a list as an input and return its reverse

Higher Order Functions

A Higher Order Functions is a function that takes a function as an argument, or returns a function as output.

Three important Higher Order Functions:

- **MAP**
  - applies a function to each element of a sequence and returns a sequence as a result.
- **REDUCE:**
  - applies a binary operation successively to the elements of a sequence.
- **FILTER:**
  - extracts elements from an sequence for which a **function** returns True .

Mapping with Python...

```
def dbl(x):
  """returns 2 * x"""
  return 2 * x

>>> map(dbl, [0, 1, 2, 3, 4])
[0, 2, 4, 6, 8]

def evens(n):
  myList = range(n)
  doubled = map(dbl, myList)
  return doubled
```

Alternatively….

```
def evens(n):
  return map(dbl, range(n))
```

Mapping with Python...

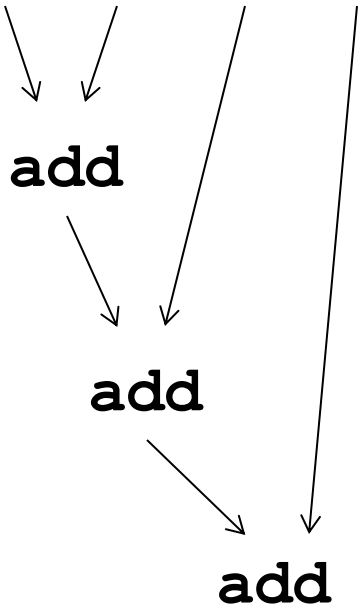
Exercise: Compute the list of the first n even numbers

`reduce`-ing with Python...

```
def add(x, y):
  """returns x + y"""
  return x + y

>>> reduce(add, [1, 2, 3, 4])
10
```

`reduce`-ing with Python...

```
def add(x, y):
  """returns x + y"""
  return x + y

>>> reduce(add, [1, 2, 3, 4])
```



Try this…

Write a function called `span` that returns the difference between the maximum and minimum numbers in a list…

```
>>> span([3, 1, 42, 7])
41
>>> span([42, 42, 42, 42])
0
```

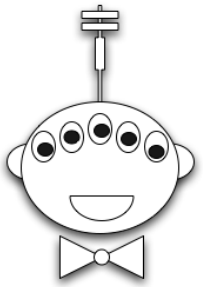
```
min(x, y)
max(x, y)
```

These are built-in to Python!

Try this...

1. Write a python function called `gauss` that takes as input a positive integer N and returns the sum
   $1 + 2 + \ldots + N$

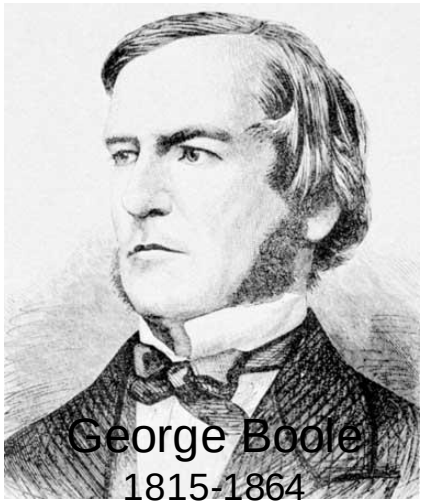
2. Write a python function called `sumOfSquares` that takes as input a positive integer N and returns the sum
   $1^2 + 2^2 + 3^2 + \ldots + N^2$



You can write extra "helper" functions too!

Booleans

Python boolean type is one of the built-in data types provided by Python, which represents **one of the two values i.e. True or False**.


George Boole
1815-1864

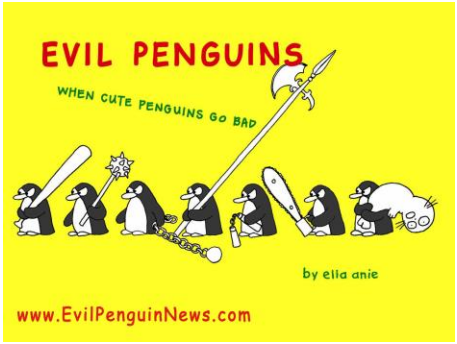
```
>>> 3 == 1+2
True
>>> 42 == "spam"
False
>>> “spam” > 42
True (in Python 2.x)
```

Strings > lists > numbers

The “Truth” about Python’s Booleans

```
>>> True + 41
42
>>> 2 ** False == True
True
```

Demonstrating the True “power” of Falsity!



if, elif, else…

```
#if elif else explanation

def special(x):
      """This function demonstrates the use
      of if and else"""
      if x == 42:
            return "Very special number!"
      else:
            return "Stupid, boring number."
```

```
def special(x):
      if x == 42:
            return "Very special number!"
      return "Stupid, boring number."
```

Alternatively??

Notice how lines with the same level of indentation are in the same code block!

if, elif, else…

```
def superSpecial(x):
    """This function demonstrates the use
    of if, elif, and else"""
    if x < 42:
        return "Small number"
    elif x == 42 or x % 42 == 0:
        return "Nice!"
    elif 41 <= x <= 43:
        return "So close!"
    else:
        # We might do more stuff here…
        return "Yuck!"
```

Would swapping the order of these elif's give the same behavior?

Notice how lines with the same level of indentation are in the same code block!

Exercises

Exercise 1

Write a function named abs that takes as an input an integer and returns its absolute value

Exercise 2

A palindrome is a word, phrase, number, or other sequence of characters which reads the same backward or forward e.g EYE, or RACECAR. Write a function named isPalindrome is a word is a palindrome or not.

Filtering

1. First input argument is a one-argument function that returns a Boolen result.
2. Second input argument is a list or sequence
3. Returns all elements in the list that makes the function True

```
def even(x):
  """returns True is x is even
     returns False otherwise """
  return x%2

>>> reduce(even, [1, 2, 3, 4])
```

Filtering Exercise

Write a function names rightTrianglesCount that takes as an input a nested list where each inner list contains three sides of a triangle in ascending order and returns the number of right triangles. For example l = [[1, 2, 3], [3, 4, 5], [1, 1, $\sqrt{2}$]], rightTrianglesCount([[1, 2, 3], [3, 4, 5], [1, 1, $\sqrt{2}$]]) should return 2.