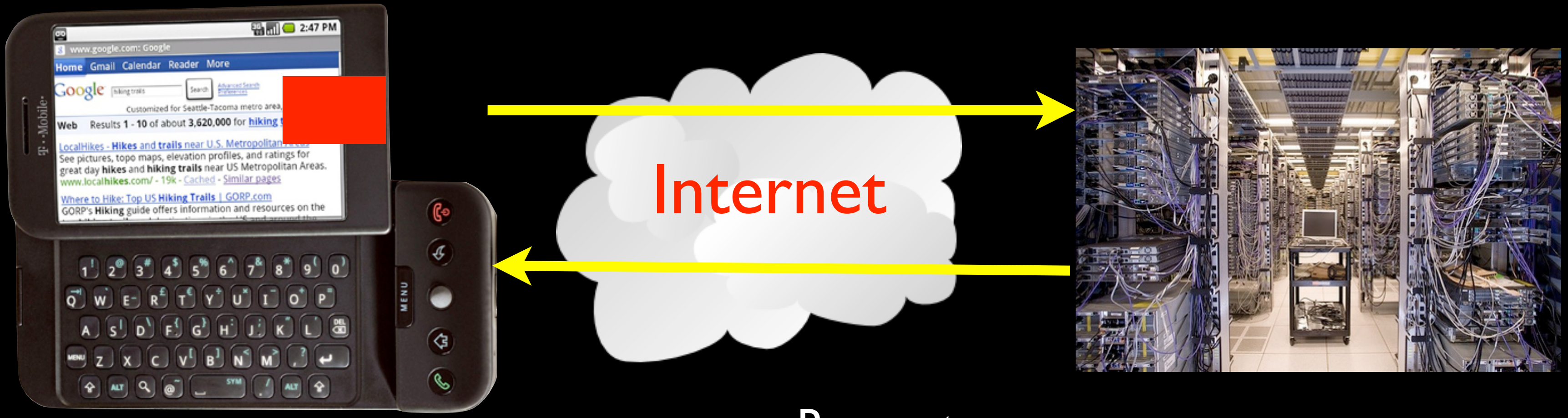
Chapter 3
Writing Simple Programs

Charles Severance

Textbook: Using Google App Engine, Charles Severance



HTML

JavaScript

AJAX

CSS

HTTP

Request

Response

GET

POST

Python

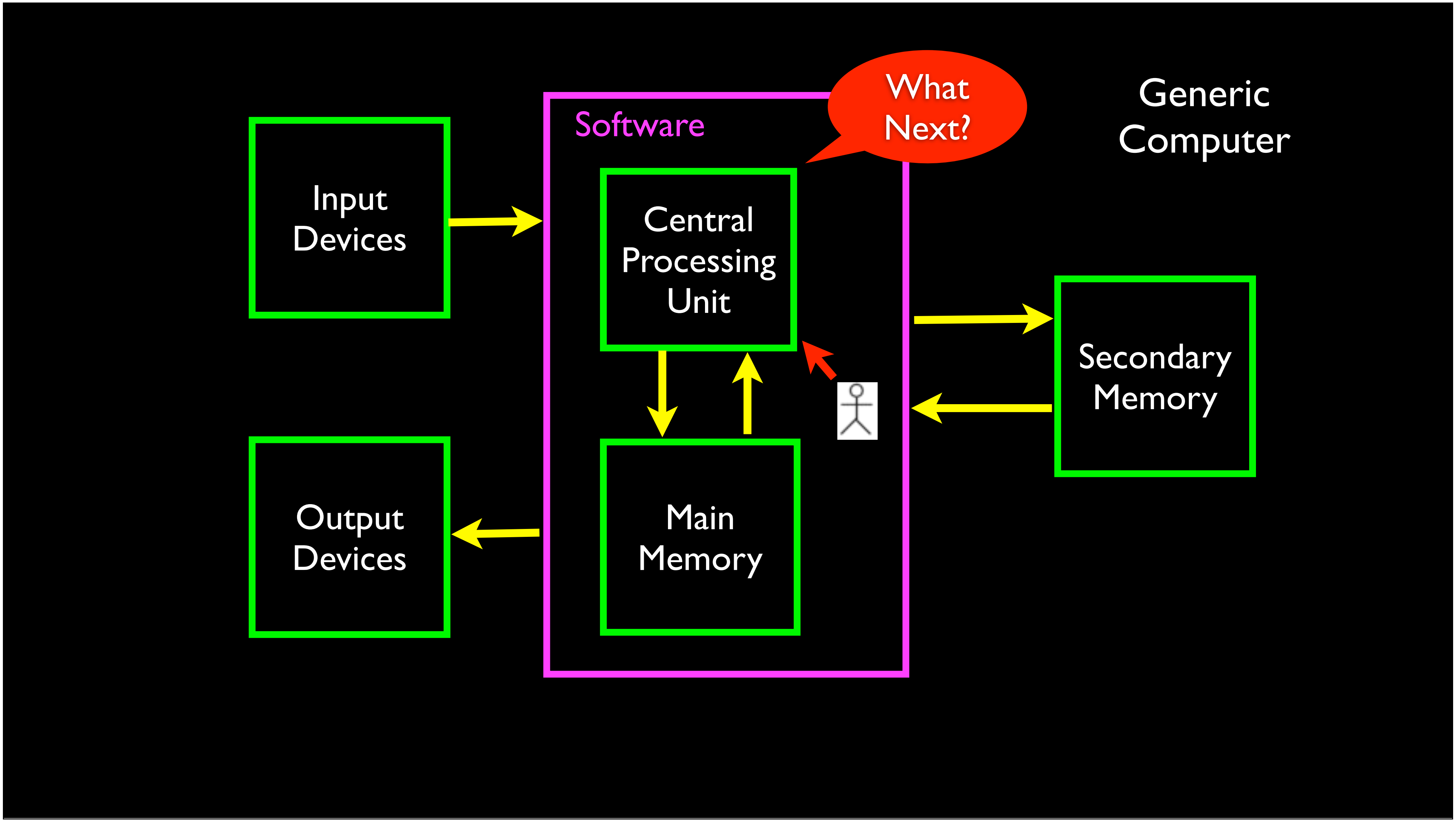
Data Store

Templates

memcache

What Is Programming?

- Witin the web server we set lots and lots of “requests” which we need to respond to
- This is the request/response cycle
- We don’t want to do this by hand
- So we write a program to communicate how we want each of the requests to be handled


Generic Computer
Input Devices
Software
Central Processing Unit
What Next?
Main Memory
Output Devices
Secondary Memory

Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a sequence of steps to be done in order
- Some steps are conditional - they may be skipped
- Sometimes a step or group of steps are to be repeated
- Sometimes we store a set of steps to be reused over and over as needed several places throughout the program

```
add 300 grams of flour
add 100 ml of milk
add an egg

if altitude > 2000:
    add an egg

add 30 grams of salt

while there are too many lumps:
     Beat mixture with a fork

open and add provided flavor packet
```

Sequential
Conditional
Repeated
Store / Reuse

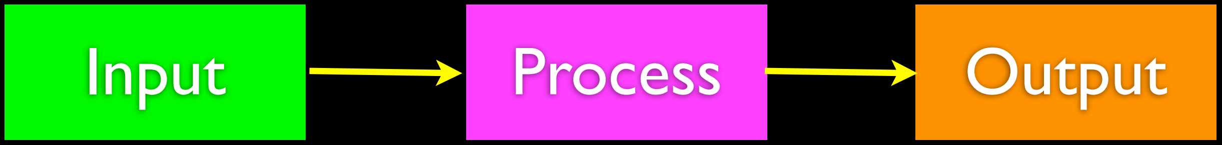
```
add 300 grams of flour
add 100 ml of milk
add an egg

if altitude > 2000:
    add an egg

    add 30 grams of salt

    while there are too many lumps:
        Beat mixture with a fork

    open and add provided flavor packet
```

Sequential
Conditional
Repeated
Store / Reuse

Elevators in Europe!

```
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```

```
python elev.py
Enter the US Floor Number: 2
Non-US Floor Number is  1
```



Input

Process

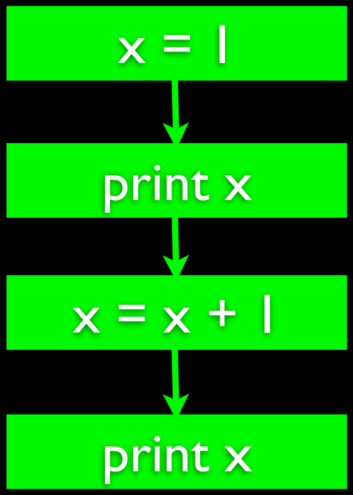
Output

The Essence of Programming

Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a sequence of steps to be done in order
- Some steps are conditional - they may be skipped
- Sometimes a step or group of steps are to be repeated
- Sometimes we store a set of steps to be reused over and over as needed several places throughout the program

Sequential Steps



Program:

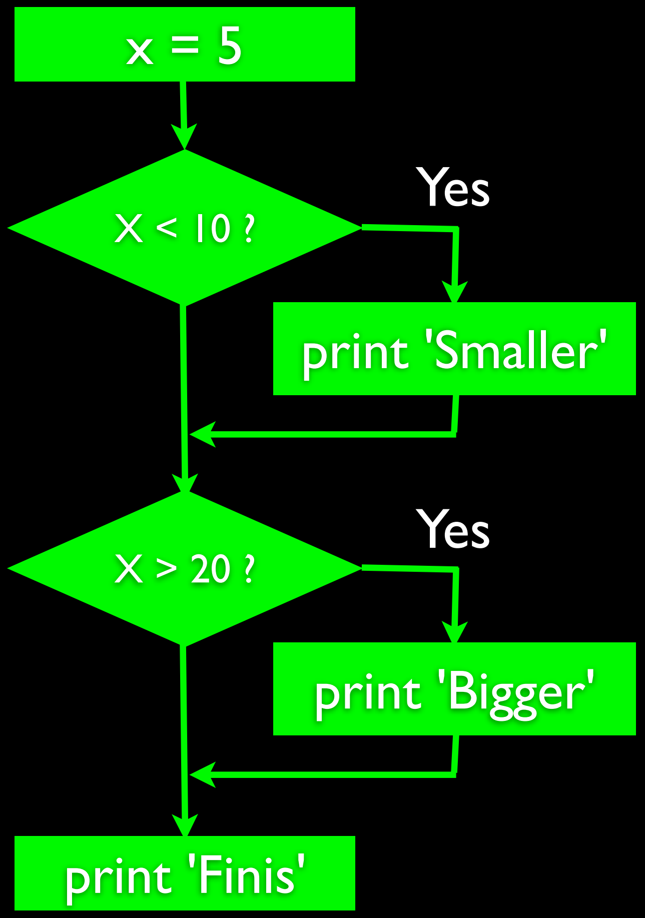
```
x = 1
print x
x = x + 1
print x
```

Output:

```
1
2
```

When a program is running, it flows from one step to the next.
We as programmers set up “paths” for the program to follow.

Conditional Steps



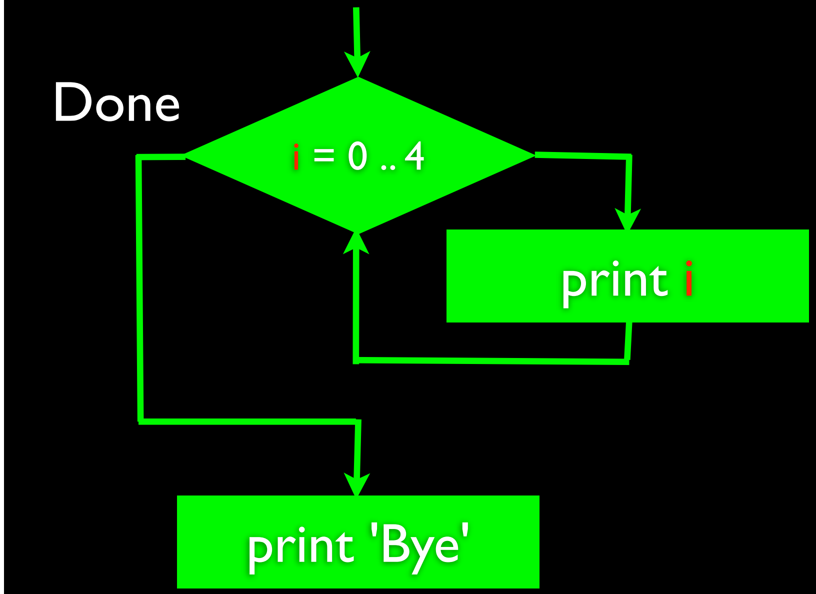
Program:

```
x = 5
if x < 10:
    print 'Smaller'

if x > 20:
    print 'Bigger'

print 'Finis'
```

Output:

```
Smaller
Finis
```

Repeated Steps



Program:

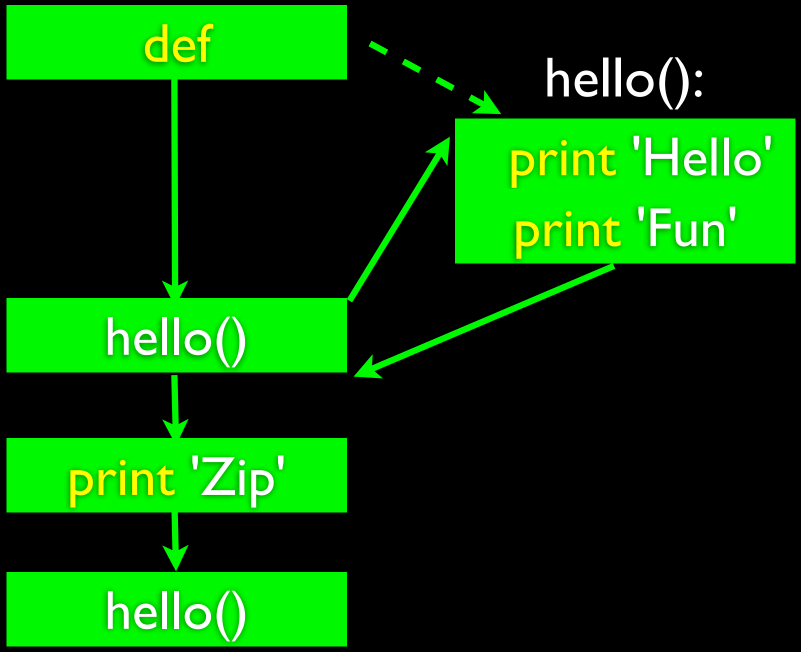
```
for i in range(5) :
    print i

print 'Bye'
```

Output:

```
0
1
2
3
4
Bye
```

Loops (repeated steps) have iteration variables that change each time through a loop. Often these iteration variables go through a sequence of numbers.

Stored (and reused) Steps



def

hello():

print 'Hello'

print 'Fun'

hello()

print 'Zip'

hello()

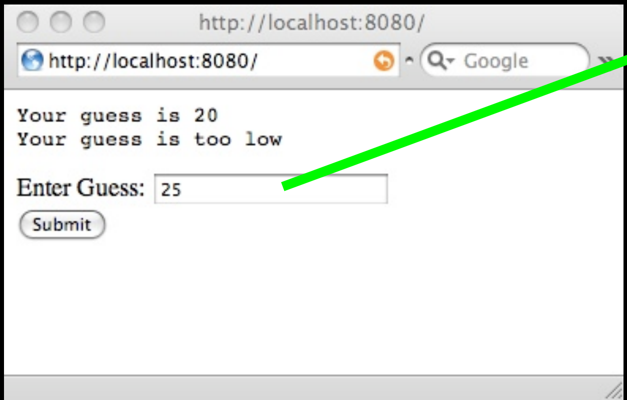
Program:

```
def hello():
    print 'Hello'
    print 'Fun'

hello()
print 'Zip'
hello()
```

Output:

```
Hello
Fun
Zip
Hello
Fun
```

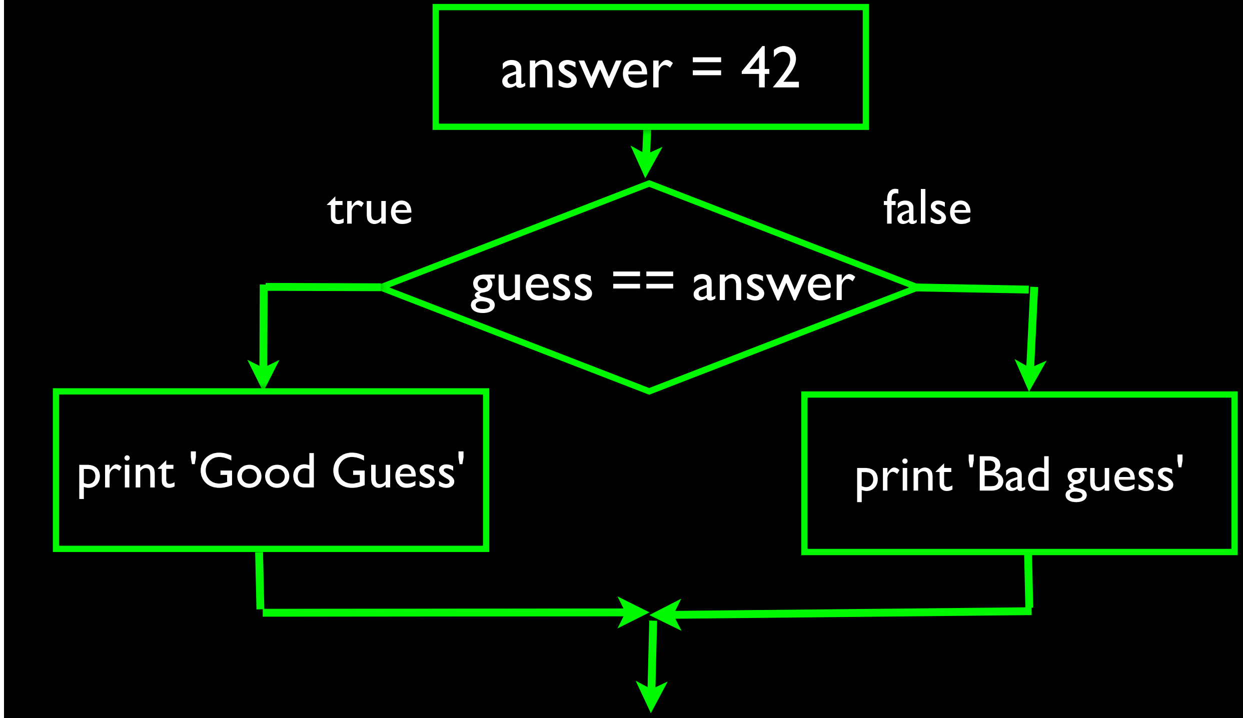
We call these little stored chunks of code "subprograms" or "functions".



```
Your guess is 20
Your guess is too low
```

Enter Guess: 25

```
print 'Your guess is', guess

answer = 42

if guess < answer :
    print 'Your guess is too low'

if guess == answer :
    print 'Congratulations!'

if guess > answer :
    print 'Your guess is too high'
```

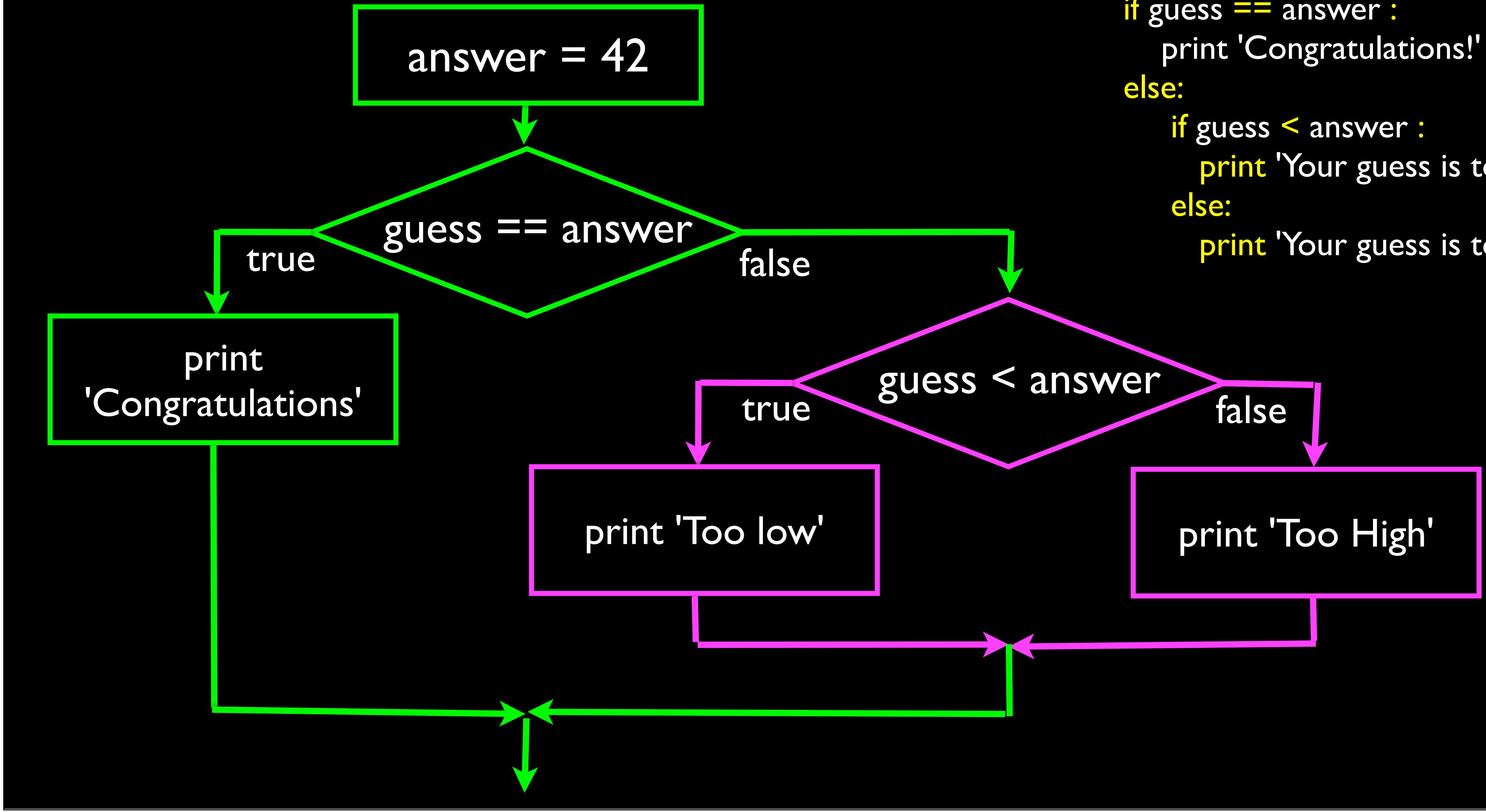


```
print 'Your guess is', guess

answer = 42

if guess == answer :
    print 'Good guess'
else:
    print 'Bad guess'
```

Nesting

- We can place a block of code within another block of code
- We call this “nesting” because the inner block is snugly nestled within the outer block

```
print 'Your guess is', guess

answer = 42

if guess == answer :
    print 'Congratulations!'
else:
    if guess < answer :
        print 'Your guess is too low'
    else:
        print 'Your guess is too high'
print 'Your guess is', guess
answer = 42
if guess == answer :
    print 'Congratulations!'
else:
     if guess < answer :
        print 'Your guess is too low'
     else:
        print 'Your guess is too high'
```



answer = 42

guess == answer

true

false

print 'Congratulations'

guess < answer

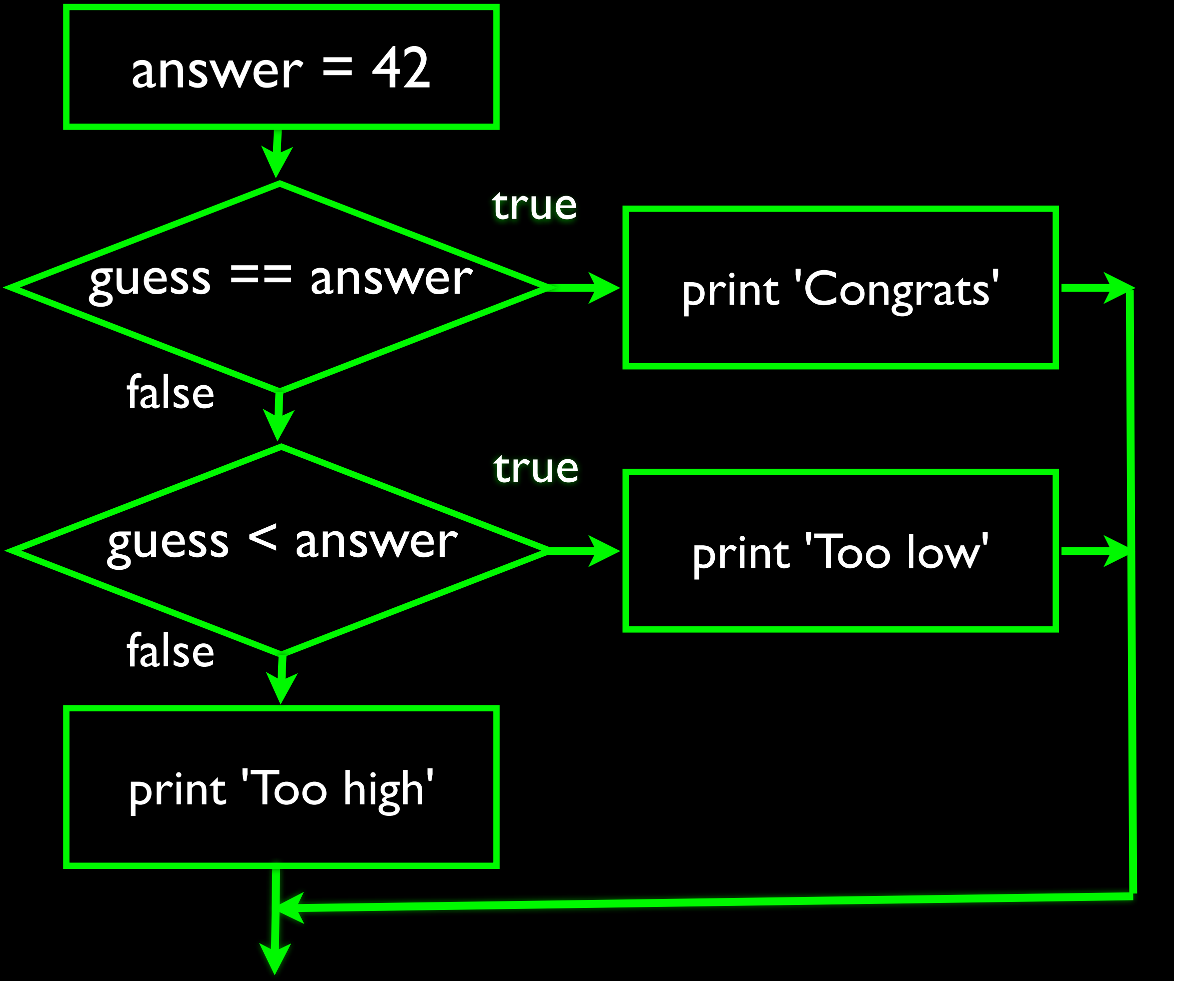
true

false

print 'Too low'

print 'Too High'

```
print 'Your guess is', guess

answer = 42

if guess == answer :
   print 'Congratulations!'
elif guess < answer :
   print 'Your guess is too low'
else:
   print 'Your guess is too high'
```



Variables

- Variables are named locations in the computer’s main memory
- We programmers use variables to store values that we want to use later in the program
- We get to pick the names of our variables (within limits)

```
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```

Rules for Python Variable Names

- Must start with a letter or underscore _
- Must consist of letters and numbers
- Case Sensitive
- Good: x usf _food food16 FOOD
- Bad: 42secret :usf value-7
- Different: usf Usf USF

Menmonic Variables

- We often try to pick mnemonic variable names to help us remember what we intended as the contents of a variable
- Non-mnemonic: x42 xyzzy snagll a123 ljkljk
- Mnemonic: count lines word usf wf

```
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```

http://en.wikipedia.org/wiki/Mnemonic

Mnemonic

```
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```

Non-Mnemonic

```
dsjdkjds = input('Enter the US Floor Number: ')
xsjdkjds = dsjdkjds - 1
print 'Non-US Floor Number is', xsjdkjds
```

Reserved Words

- You can not use reserved words as variable names / identifiers

and del for is raise
assert elif from lambda return
break else global not try
class except if or while
continue exec import pass yield
def finally in print

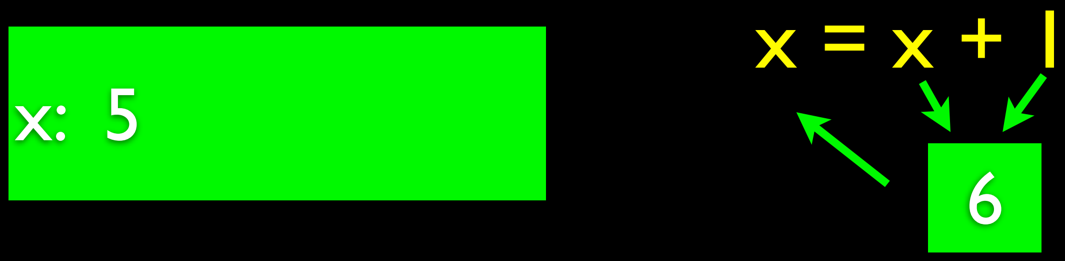
Assignment Statements

- variable = expression
- Evaluate the expression to a value and then put that value into the variable

```
x = 1
spam = 2 + 3
spam = x + 1
x = x + 1
```

Slow Motion Assignment

- We can use the same variable on the left and right side of an assignment statement
- Remember that the right side is evaluated *before* the variable is updated



x: 5

x = x + 1

6

Input Statements

- input('Prompt') - displays the prompt and waits for us to input an expression - this works for numbers
- In Chapter 4 we will see how to read strings

```
>>> x = input('Enter ')
Enter 123
>>> print x
123
```

Constants

- We use the term “constant” or “literal” to indicate a value that is not a variable

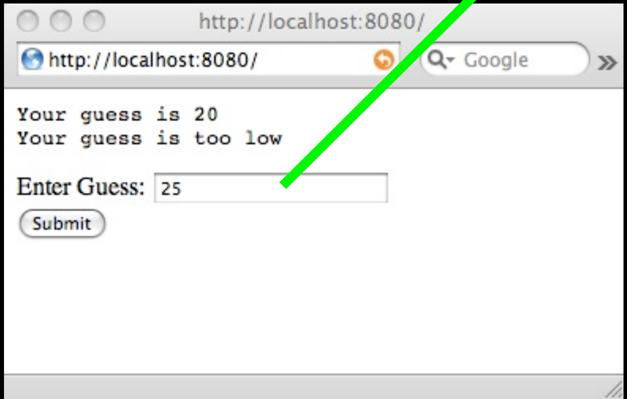
```
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is', wf
```

String Data

- Modern programs work with string data ('Fred', 'Ann Arbor', ...) far more often than numeric data (1, 2, 3.14159)
- Python has great support for working with strings

```
print 'Your guess is', guess

answer = 42

if guess == answer :
   msg = 'Congratulations!'
elif guess < answer :
   msg = 'Your guess is too low'
else:
   msg = 'Your guess is too high'

print msg
```

We can use string constants and string variables to simplify our program. Remember that a variable is a place in memory that we can use to store something that we want to use later in our program.



```
Your guess is 20
Your guess is too low
```

Enter Guess: 25

When you type "25" into the form and press "Submit", your browser sends a string like "guess=25" to your web application.

guess=25

```
print 'Your guess is', guess

answer = 42

if guess < answer :
    print 'Your guess is too low'

if guess == answer :
    print 'Congratulations!'

if guess > answer :
    print 'Your guess is too high'
```

Indexing Strings

- We can look at each character in a string by “indexing” the string with square brackets “[“ and “]”
- The first character in a string is [0]

```
python
>>> txt = 'guess=25'
>>> print txt[0]
g
>>> print txt[1]
u
>>>
```

```
guess=25
01234567
```

Slicing Strings

- We can extract a range of characters in a string using two numbers and a colon (:) in the square brackets
- The second value means "up to but not including"

```
guess=25
01234567
```

```
python
>>> txt = 'guess=25'
>>> print txt[2:4]
es
>>> print txt[2:5]
ess
>>>
```

Slicing Strings

- When we slice strings, we can omit the first or second number and it implies beginning and end of the string respectively

```
guess=25
01234567
```

```
python
>>> txt = 'guess=25'
>>> print txt[:5]
guess
>>> print txt[6:]
25
>>>
```

Concatenating Strings

- We use the “+” operator to concatenate two strings
- If we want a space between the strings, we need to add the space

```
>>> think = 'happy' + 'thoughts'
>>> print think
happythoughts
>>>
```

```
>>> think = 'happy' + '  ' + 'thoughts'
>>> print think
happy thoughts
>>>
```

The Python String Library

- Python has a number of powerful string manipulation capabilities in the string library (an example of the store and reuse pattern)

```
>>> txt = 'guess=25'
>>> print txt.find('=')
5
>>> print txt.find('pizza')
-1
>>>
```

```
guess=25
01234567
```

The Python String Library

- Other capabilities in the string library include : lowercase(), rfind(), split(), strip(), rstrip(), replace(), and many more

http://docs.python.org/library/stdtypes.html

Types and Conversion

- Every variable and constant in Python as a “type” and Python knows the type
- If you do something that is not allowed for a particular type, you will get an error
- We can ask Python which type a variable is using the built-in type() function

```
>>> txt = 'guess=25'
>>> print type(txt)
<type 'str'>
>>> pos = txt.find('=')
>>> print pos
5
>>> print type(pos)
<type 'int'>
>>>
```

Breaking the Rules...

```
>>> txt = 'abc' + 'def'
>>> print txt
abcdef
>>> num = 36 + 6
>>> print num
42
>>> huh = 'abc' + 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
>>>
```

Converting Integer to String

```
>>> txt = 'abc' + 'def'
>>> print txt
abcdef
>>> num = 36 + 6
>>> print num
42
>>> huh = 'abc' + str(6)
>>> print huh
abc6
>>>
```

If we convert the integer to a string using the built-in str() function - the types match for the concatenation (+) operation.

Parsing a String

- Sometimes we want to break a string into pieces and do something with those pieces in several steps

```
guess=25
01234567
```

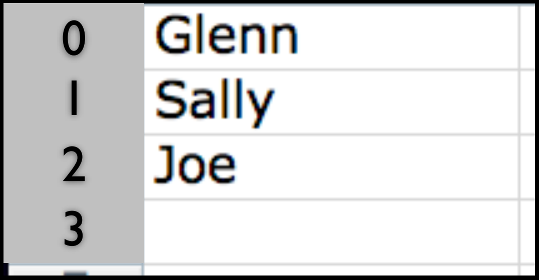
```
>>> txt = 'guess=25'
>>> pos = txt.find('=')
>>> sub = txt[pos+1:]
>>> print sub
25
>>> print type(sub)
<type 'str'>
>>> ival = int(sub)
>>> print ival
25
>>> print type(ival)
<type 'int'>
>>>
```

Multi-Value Variables: Collections

- So far all variables we ave seen contain a single value and if we put a new value into the variable, it over-writes the existing value
- Collections are a type of variable that can contain more than one value at the same time
- We need a way to organize, store, and retrieve these multiple values

A Python List Object

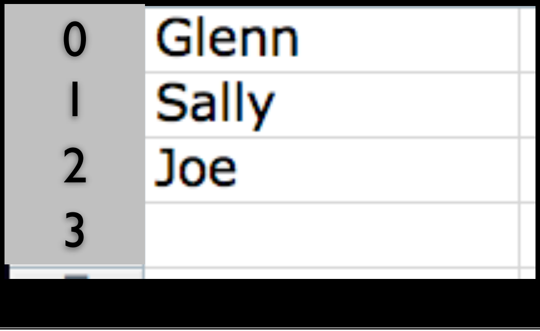
- A Python list() object can contain more than one item
- Lists are stored in order and are indexed by the position of a value within a list
- They are like a one column spreadsheet
- Python lists follow "European Elevator" rules



0	Glenn
1	Sally
2	Joe
3	

```
>>> pals = list()
>>> pals.append('Glenn')
>>> pals.append('Sally')
>>> pals.append('Joe')
>>> print pals
['Glenn', 'Sally', 'Joe']
>>> print type(pals)
<type 'list'>
>>> print len(pals)
3
>>> print pals[0]
Glenn
>>> print pals[2]
Joe
>>>
```

- We create an empty list by calling the built-in function list()
- We add new elements using append()
- We can find the length of the list using the built in function len()
- We can index the list with square brackets



0	Glenn
1	Sally
2	Joe
3	

```
>>> print pals
['Glenn', 'Sally', 'Joe']
>>> pals[2] = 'Joseph'
>>> print pals
['Glenn', 'Sally', 'Joseph']
>>>
>>> pals.sort()
>>> print pals
['Glenn', 'Joseph', 'Sally']
>>>
```

- We can replace an element in a list by using an index in an assignment statement
- We can sort a list using the sort method in the list library

Looping Through Lists

- Loops are an example of the “repeated code” pattern
- We construct a loop using for and in will execute a block of code once for each value in the list
- We define an iteration variable that takes on the successive elemnents of the list each time through the loop

```
pals = ['Glenn', 'Sally', 'Joseph']

for x in pals:
    print x

print 'Done'
```

Looping Through Lists

- Loops are an example of the "repeated code" pattern
- We construct a loop using for and in will execute a block of code once for each value in the list
- We define an iteration variable that takes on the successive elemnents of the list each time through the loop

```
>>> print pals
['Glenn', 'Sally', 'Joseph']
>>> for x in pals:
...         print x
...
Glenn
Sally
Joseph
>>>
```

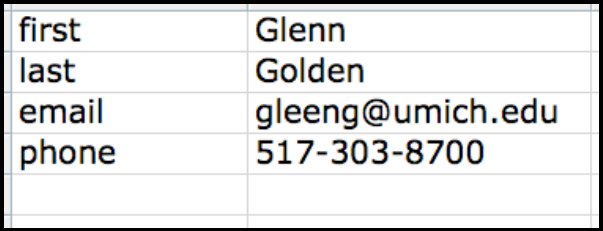
Looping Through Strings

- Strings function very much like a “list of characters”
- So we can construct a loop using for and in will execute a block of code once for each value in the list
- We define an iteration variable that takes on the successive elemnents of the list each time through the loop

```
>>> txt = 'guess=25'
>>> for x in txt:
...          print x
...
g
u
e
s
s
=
2
5
>>>
```

Python's Backpack: Dictionaries

- Sometimes we want a bunch of stuff in a collection where each item has a label and the label allows us to store and retrieve a value
- It is more like a two-column spreadsheet



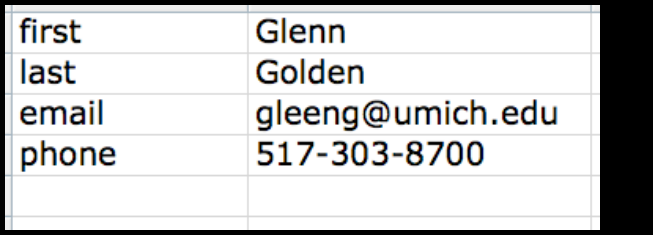
label	value
first	Glenn
last	Golden
email	gleeng@umich.edu
phone	517-303-8700

- We create an empty dictionary by calling dict()

- We fill up our dictionary with assignment statements where the index is the “key” or “label” which marks a value

- When we print a dictionary we see a list of mappings of a key to a value

```
>>> pal = dict()
>>> pal['first'] = 'Glenn'
>>> pal['last'] = 'Golden'
>>> pal['email'] = 'glenng@umich.edu'
>>> pal['phone'] = '517-303-8700'
>>> print pal
{'phone': '517-303-8700', 'last':
'Golden', 'email': 'glenng@umich.edu',
'first': 'Glenn'}
>>>
```

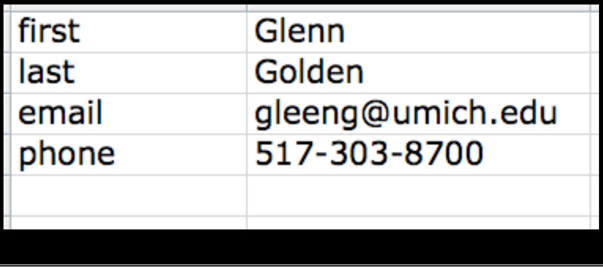


label	value
first	Glenn
last	Golden
email	gleeng@umich.edu
phone	517-303-8700

Retrieving Data

- To retrieve an value from the dictionary, we can use the index operation “[“
- But we must make sure that the key exists

label value



first	Glenn
last	Golden
email	gleeng@umich.edu
phone	517-303-8700

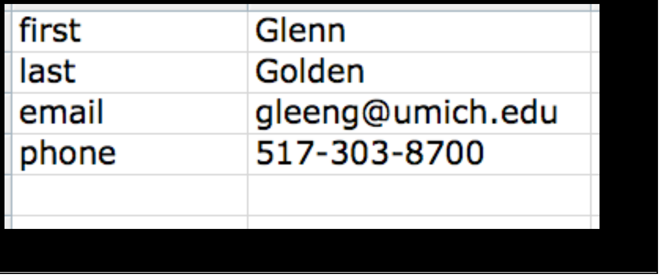
```
>>> print pal
{'phone': '517-303-8700', 'last':
'Golden', 'email': 'glenng@umich.edu',
'first': 'Glenn'}
>>> print pal['phone']
517-303-8700
>>>
>>> print pal['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
>>>
```

Safe Dictionary Retrieval

- We can deal with the missing key by using the built-in function get() and specifying a value to return if the key is not found

```
>>> print pal
{'phone': '517-303-8700', 'last': 'Golden', 'email': 'glenng@umich.edu',
'first': 'Glenn'}
>>> print pal.get('age','Age not available')
Age not available
>>> print pal.get('phone', 'Phone not available')
517-303-8700
>>>
```

label value



first	Glenn
last	Golden
email	gleeng@umich.edu
phone	517-303-8700

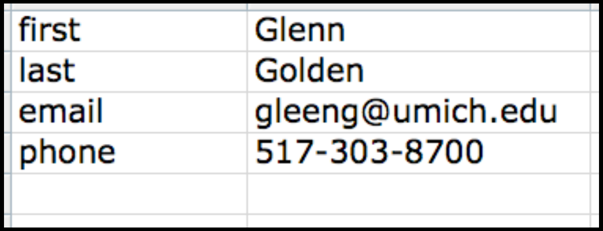
Looping Through a Dictionary

- Dictionaries do not maintain order - but we can loop through them
- We construct a loop with for and in
- The iteration variable loops through the keys in the dictionary

```
>>> print pal
{'phone': '517-303-8700',
'last': 'Golden', 'email':
'glenng@umich.edu', 'first':
'Glenn'}
>>> for z in pal:
...          print z
...
phone
last
email
first
>>>
```

Looping Through a Dictionary

- If we can loop through the keys, we can then use those keys to look up the values



key	value
first	Glenn
last	Golden
email	gleeng@umich.edu
phone	517-303-8700

```
>>> print pal
{'phone': '517-303-8700',
'last': 'Golden', 'email':
'glenng@umich.edu', 'first':
'Glenn'}
>>> for key in pal:
...         print key, pal[key]
...
phone 517-303-8700
last Golden
email glenng@umich.edu
first Glenn
>>>
```

Store/Reuse: Functions

- In the last pattern, we want to write code once and reuse it several places - this allows us to make changes one place and allows us to organize larger programs into logical sub-units

Function: How Long is a Sequence?

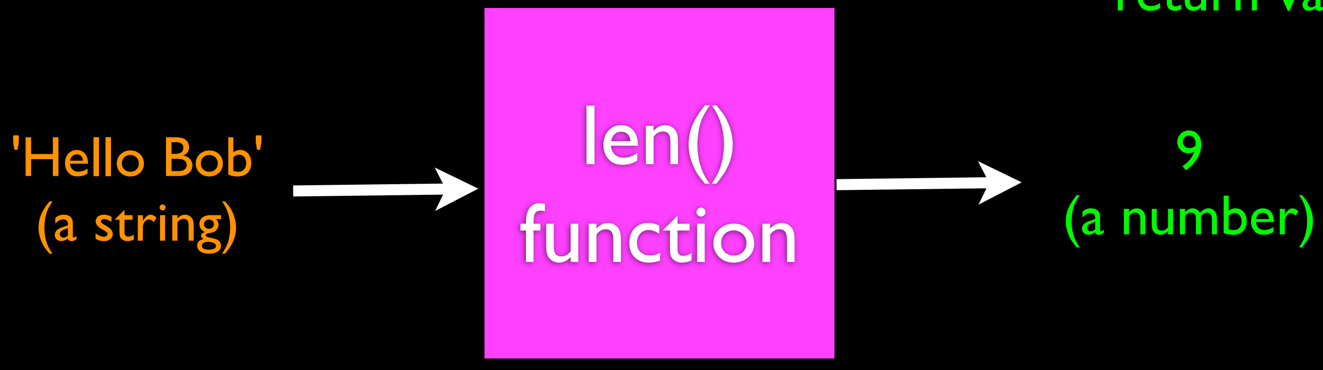
- The len() function takes a string as a parameter and returns the number of characters in the string
- Actually len() tells us the number of elements of any set or sequence

```
>>> greet = 'Hello Bob'
>>> print len(greet)
9
>>> x = [ 1, 2, 'fred', 99]
>>> print len(x)
4
>>>
```

Len Function

```
>>> greet = 'Hello Bob'
>>> x = len(greet)
>>> print x
9
```

A function is some stored code that we use. A function takes some input parameter(s) and produces an output return value.



'Hello Bob'
(a string)

len()
function

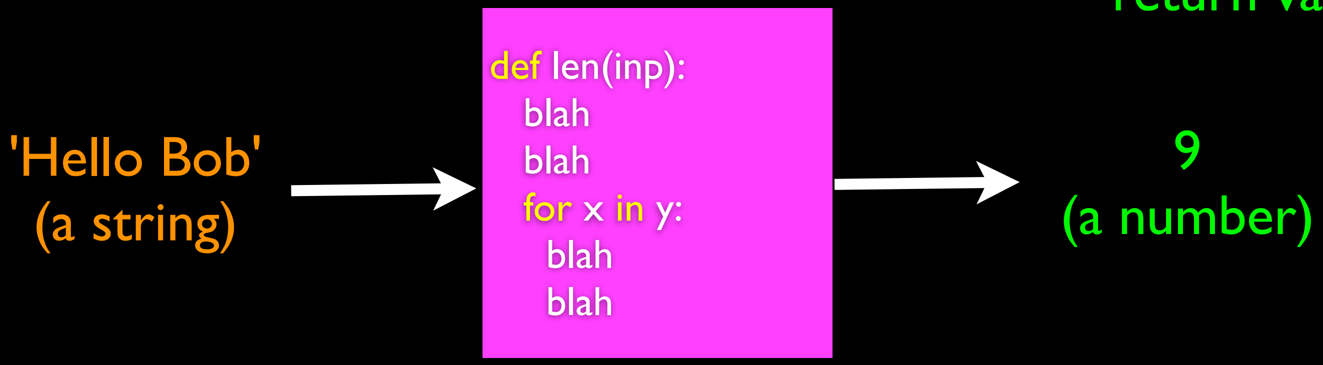
9
(a number)

Guido wrote this code

Len Function

```
>>> greet = 'Hello Bob'
>>> x = len(greet)
>>> print x
9
```

A function is some stored code that we use. A function takes some input parameter(s) and produces an output return value.



'Hello Bob'
(a string)

```
def len(inp):
   blah
   blah
   for x in y:
     blah
     blah
```

9
(a number)

A Trivial Function

- The def keyword indicates the beginning of a block and defines a name for the function

- The block of code is not executed as part of the "def" process

- We can execute the code later using the name we assigned to the function.

```
>>> def welcome():
...         print 'Hello'
...
>>> welcome()
Hello
>>> welcome()
Hello
>>>
```

Parameters to Functions

- Sometimes we want to feed the function some value (i.e. a parameter) as its input so we can use the function for different purposes

- We define the "formal parameter" on the def statement

- We pass in the "actual parameter" on the function call

```
>>> def welcome(name):
...         print 'Hello',name
...
>>> welcome('Glenn')
Hello Glenn
>>> welcome('Sally')
Hello Sally
>>>
```

Return Values

- A function can “return” a value back to its caller using the “return” statement

- This return value “comes back” and can be used in an assignment statement or expression

```
>>> def greet(lang):
...     if lang == 'es':
...         return 'Hola'
...     elif lang == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...
>>> xgr = greet('fr')
>>> print xgr, ’Michael'
Bonjour Michael
>>>
```

Return Values

- A function can “return” a value back to its caller using the “return” statement
- This return value “comes back” and can be used in an assignment statement or expression

```
>>> def greet(lang):
...     if lang == 'es':
...         return 'Hola'
...     elif lang == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...
>>> print greet('en'),'Glenn'
Hello Glenn
>>> print greet('es'),'Sally'
Hola Sally
>>> print greet('fr'),'Michael'
Bonjour Michael
>>>
```

The try / except Structure

- Sometimes we know a line of code may cause traceback - perhaps because it is processing user input that may be flawed
- You surround a dangerous section of code with try and except.
- If the code in the try works - the except is skipped
- If the code in the try fails - it jumps to the except section

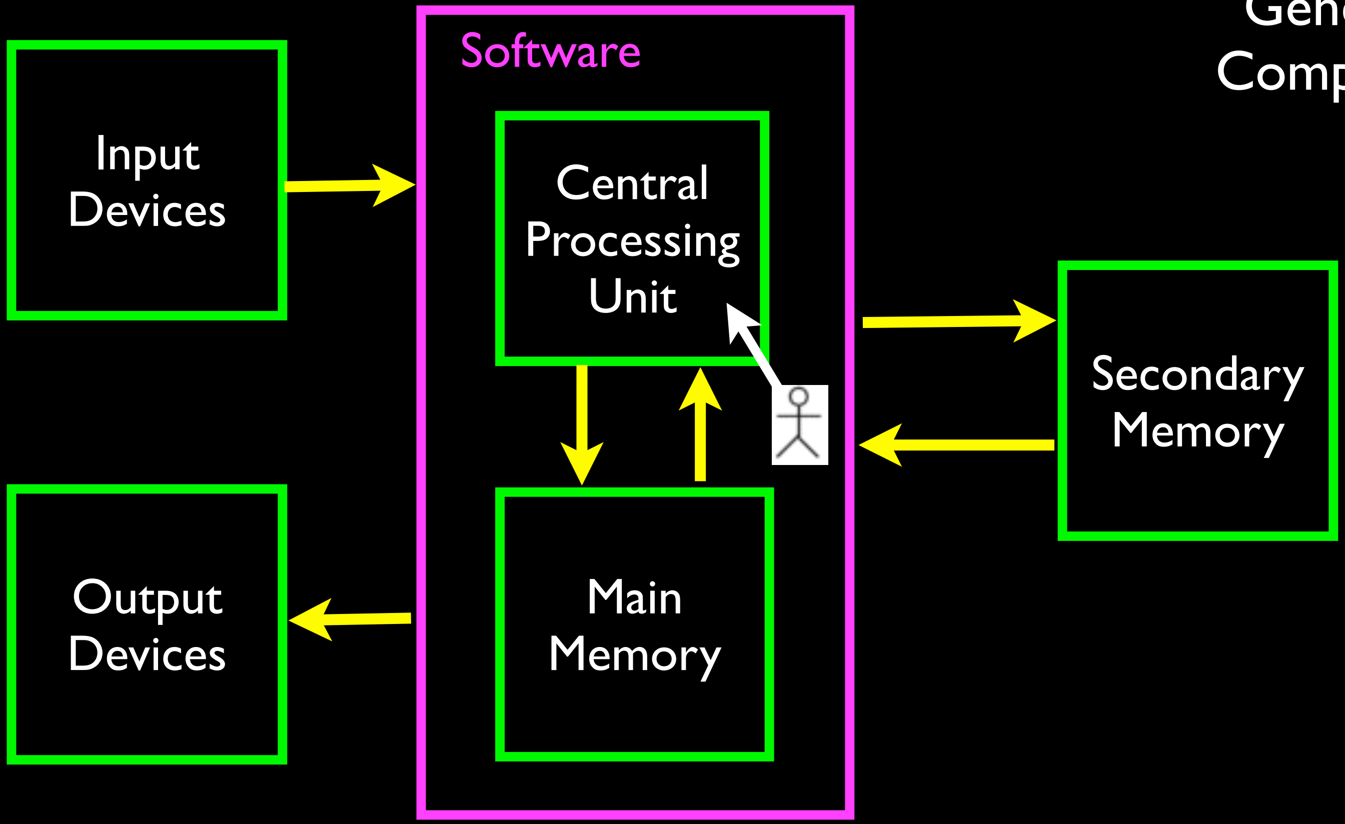
```
$ cat notry.py
astr = 'fourtytwo'
istr = int(astr)
```

The program stops here

```
$ python notry.py
Traceback (most recent call last):
  File "notry.py", line 6, in <module>
    istr = int(astr)
ValueError: invalid literal for int() with base 10: 'fourtytewo'
```

All Done

Generic Computer



Software

Input Devices

Central Processing Unit

Main Memory

Secondary Memory

Output Devices

```
$ cat tryexcept.py
astr = 'fourtytwo'
try:
    istr = int(astr)
except:
    istr = -1

print 'First', istr

astr = '42'
try:
    istr = int(astr)
except:
    istr = -1

print 'Second', istr
```

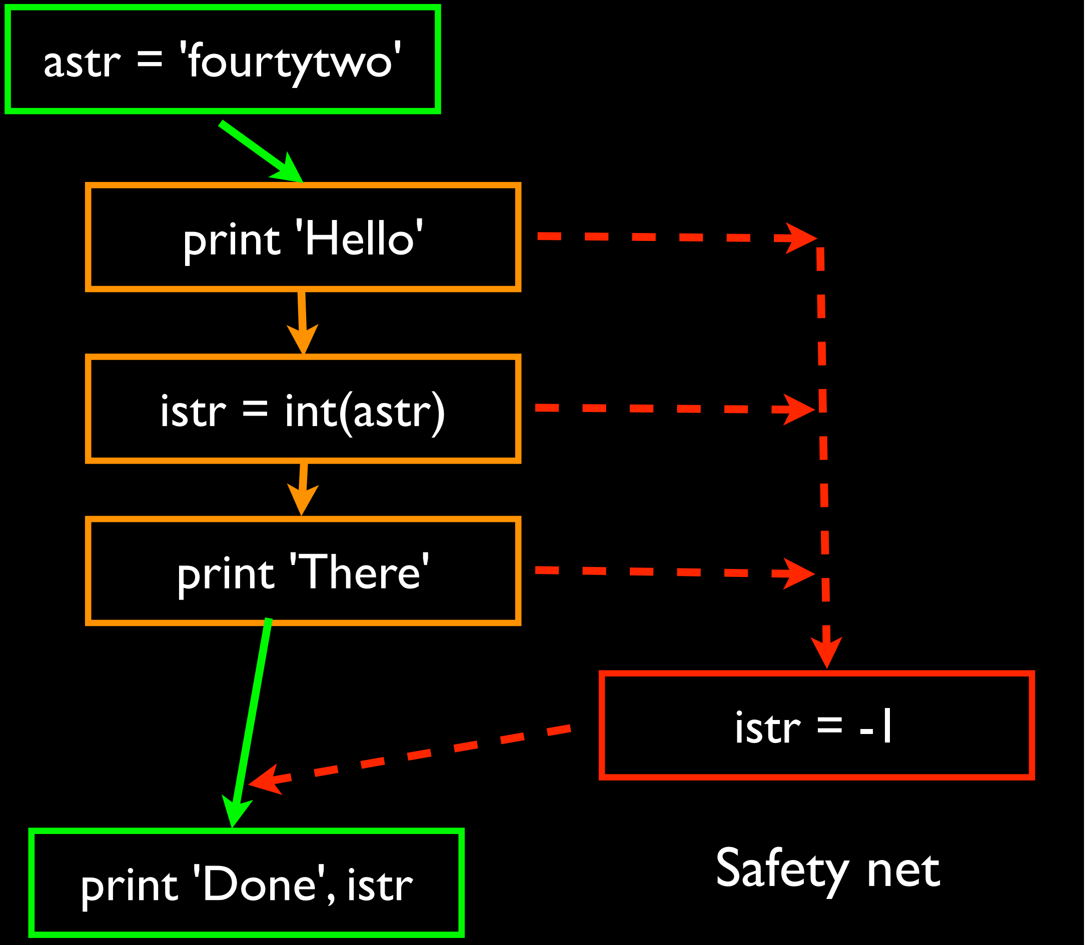
When the first conversion fails - it just drops into the except: clause and the program continues.

```
$ python tryexcept.py
First -1
Second 42
```

When the second conversion succeeds - it just skips the except: clause and the program continues.

try / except

```
astr = 'fourtytwo'
try:
    print 'Hello'
    istr = int(astr)
    print 'There'
except:
    istr = -1

print 'Done', istr
```



astr = 'fourtytwo'

print 'Hello'

istr = int(astr)

print 'There'

istr = -1

print 'Done', istr

Safety net

Comments in Python

- Python ignores blank lines in Python programs and ignores everything after it sees “#” on a line

```
# This program helps travelers use elevators
usf = input('Enter the US Floor Number: ')
wf = usf – 1  # The conversion is quite simple
print 'Non-US Floor Number is',wf
```

Comments in Python

- Python ignores blank lines in Python programs and ignores everything after it sees “#” on a line

```
usf = input('Enter the US Floor Number: ')
wf = usf – 1
print 'Non-US Floor Number is',wf
```

Questions…