Defining Functions

Chapter 6

Why functions?

- Functions allow decomposition of programs into manageable, reusable, components
- Functions are similar to mathematical functions, but also differ:
  - parameter "passing"
  - side effects

The function of functions

- What you've already seen:
  - Programs having a "`main`" function
  - Built-in Python functions: `abs`, `min`, `max`,...
    - http://docs.python.org/lib/built-in-funcs.html
  - Library functions: `math.sqrt`, ...
- Functions capture "canned" computations that can be reused

The function of functions

- Functions avoid drawbacks of duplicating similar/identical code in different places:
  - writing same code twice
  - maintaining separate copies
- Functions make code more readable

Functions, informally

- A function is like a subprogram: a small program inside a larger program
- A function gives a name to that subprogram
- The subprogram can be executed by referring to the name

Functions, informally

- Functions are first defined
- The definition can then be used: "called"/"invoked"

Functions, informally

```
def main():
  print “Happy birthday to you!”
  print “Happy birthday to you!”
  print “Happy birthday, dear Fred..."
  print “Happy birthday to you!”
```


Functions, informally

```
def main():
  print “Happy birthday to you!”
  print “Happy birthday to you!”
  print “Happy birthday, dear Fred..."
  print “Happy birthday to you!”
```

Functions, informally

```
def happy():
  print “Happy birthday to you!”
def main():
  print “Happy birthday to you!”
  print “Happy birthday to you!”
  print “Happy birthday, dear Fred..."
  print “Happy birthday to you!”
```


Functions, informally

```
def happy():
  print “Happy birthday to you!”
def main():
  happy()
  happy()
  print “Happy birthday, dear Fred..."
  happy()
```

Functions, informally

```
def happy():
  print “Happy birthday to you!”
def main():
  happy()
  happy()
  print “Happy birthday, dear Fred..."
  happy()
```


Functions, informally

```
def happy():
  print “Happy birthday to you!”
def singfred():
  happy()
  happy()
  print “Happy birthday, dear Fred..."
  happy()
```

Functions, informally

```
def happy():
  print “Happy birthday to you!”
def singfred():
  happy()
  happy()
  print “Happy birthday, dear Fred..."
  happy()
```


Functions, informally

```
def happy():
  print “Happy birthday to you!”
def singfred():
  happy()
  happy()
  print “Happy birthday, dear Fred..."
  happy()
```

Functions, informally

```
def happy():
  print “Happy birthday to you!”
def singfred():
  happy()
  happy()
  print “Happy birthday, dear Fred..."
  happy()
def singlucy():
  happy()
  happy()
  print “Happy birthday, dear Lucy..."
  happy()
```


Functions, informally

```
def main():
  singFred()
  print
  singLucy()
```

Functions, informally

```
def sing(person):
  happy()
  happy()
  print “Happy birthday dear”, person\
    + “...”
  happy()


def main():
  singFred()
  print
  singLucy()
```


Functions, informally

```
def sing(person):
  happy()
  happy()
  print “Happy birthday dear”, person\
    + “...”
  happy()


def main():
  singFred()
  print
  singLucy()
```

Functions, informally

```
def sing(person):
   happy()
   happy()
   print “Happy birthday dear”, person\
     + “...”
   happy()


def main():
   sing(“Fred”)
   print
   sing(“Lucy”)
```


Functions, informally

* A parameter is a variable that is initialized to the value of the corresponding argument when the function is called

Functions, informally

- Functions use "bound" variables and "unbound" variables
- This is tricky!

```
x = 10

def f(x):
 return x+1

def g():
 return x+1
```


Functions & parameters: the details

- Variables are bound to "scopes"
- Different scopes can use the same name for different variables
- Variables can be "globally" scoped
- Variables can be scoped "locally" within a function

Functions & parameters: the details

```
x = 10
def f(x):
  x = x+1
  return x
x = 11
def main():
  y=f(x)
  print x,y
x = 12
main()
```


Functions & parameters: the details

```
x = 10
def f(x):
  x = x+1
  return x
x = 11
def main():
  y=f(x)
  print x,y
x = 12
main()
```

- What does this program print out?

A. 10 11
B. 12 13
C. 11 11
D. 12 12

Functions & parameters: the details

```
x = 10
def f(x):
  x = x+1
  return x
x = 11
def main():
  y=f(x)
  print x,y
x = 12
main()
```

- What does this program print out?

A. 10 11
B. 12 13
C. 11 11
D. 12 12

Functions & parameters: the details

- Parameters, like all variables defined within a function, are only accessible in the body of their function
- Variables with the same names elsewhere in the program are distinct from parameters and variables defined in a function

Functions & parameters: the details

- A function is defined with a possibly empty list of parameters:

```
def name(parameters):
  body
```

- A function is called by using its name followed by a list of arguments, one for each parameter:

```
name(arguments)
```


Functions & parameters: the details

- At a call:
  - Execution of the caller is suspended
  - The arguments to the call are assigned to the parameters of the called function
  - The body of the called function is executed
  - Execution of the caller is resumed

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```


Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Fred"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```


Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Fred"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Fred"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

```
def happy():
    print "Happy birthday to you!"
```


Functions & parameters: the details

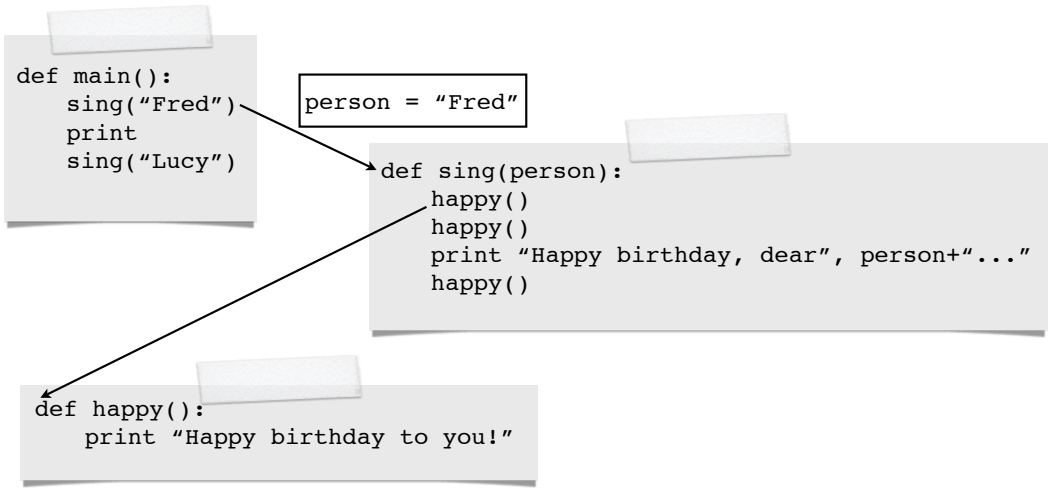
```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Fred"

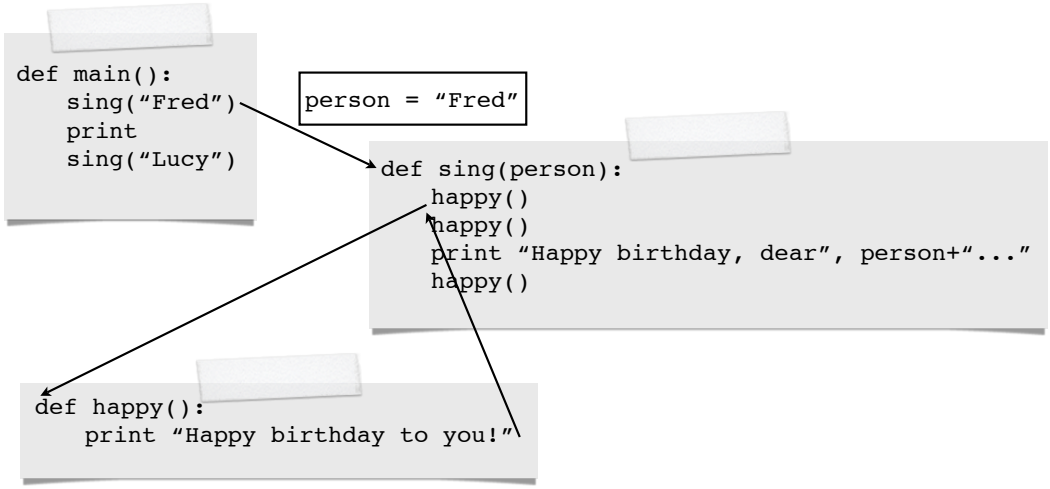
```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

```
def happy():
    print "Happy birthday to you!"
```

Functions & parameters: the details



Functions & parameters: the details



Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Fred"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

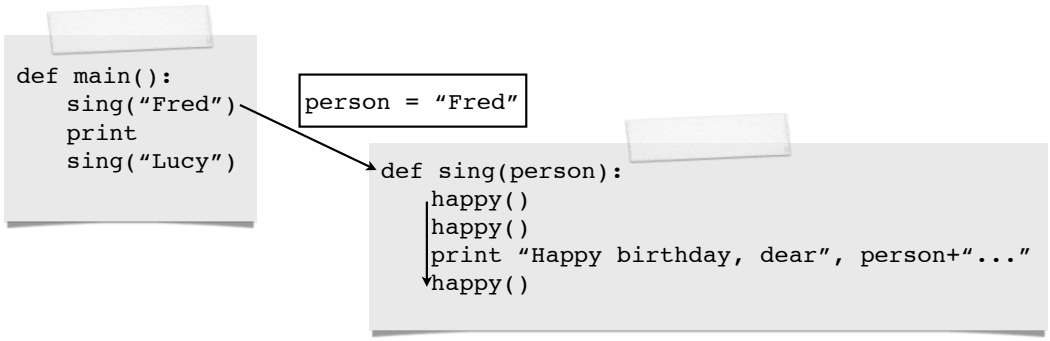

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

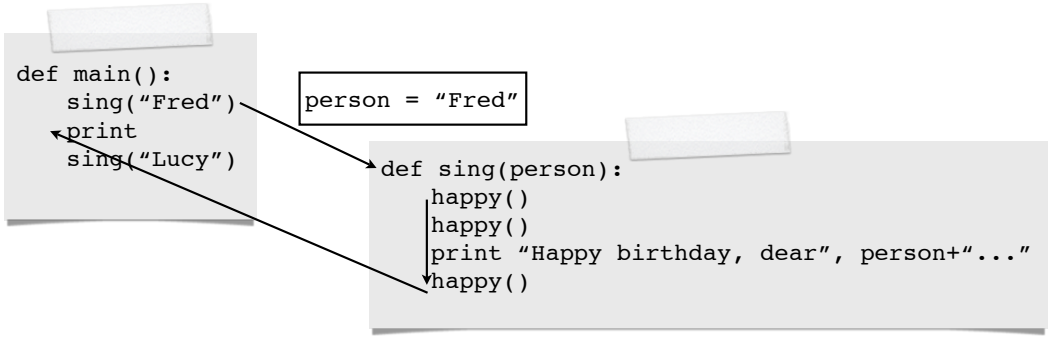
person = "Fred"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

Functions & parameters: the details



Functions & parameters: the details



Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```


Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```


Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

```
person = "Lucy"
```

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Lucy"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```


Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Lucy"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

person = "Lucy"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

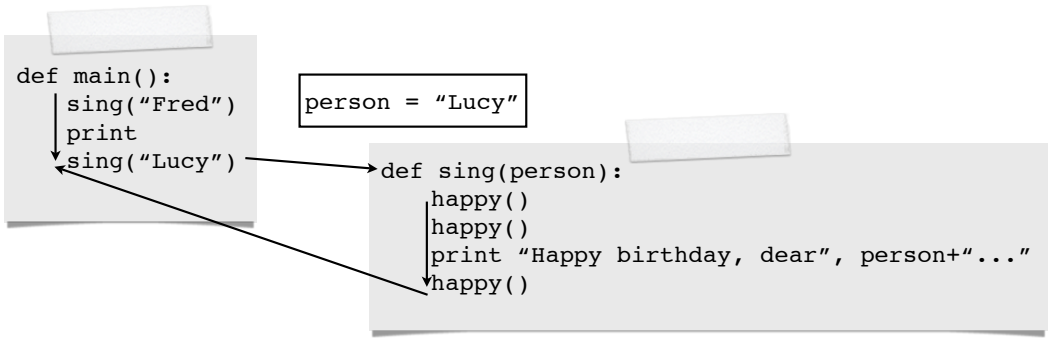

Functions & parameters: the details

```
def main():
    sing("Fred")
    print
    sing("Lucy")
```

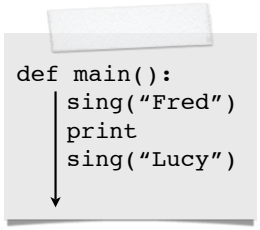
person = "Lucy"

```
def sing(person):
    happy()
    happy()
    print "Happy birthday, dear", person+"..."
    happy()
```

Functions & parameters: the details



Functions & parameters: the details



Functions & parameters: the details

- Multiple parameters are matched up by position: arguments are assigned one by one to their parameters in the order they appear
- Consider:

```
def f(x,y): return x+y
```

- `f(1,2)` assigns `1` to `x` then `2` to `y`

Getting results from functions

- Arguments are inputs to functions, initializing their parameters
- Calling the same function with different arguments can produce different results
- We've already seen use of the `return` statement to provide a result

Example

```
def square(x):
   return x*x
def distance(p1, p2):
   d = math.sqrt(
      square(p2[0] - p1[0]) + square(p2[1] - p1[1]))
   return d
```


Getting results from functions

- Functions can return multiple results
- Get multiple results from call using multiple assignment

```
def sum_diff(x,y):
   sum = x+y
   diff = x-y
   return sum, diff

s,d = sum_diff(a,b)
```

Getting results from functions

- Be careful not to forget the `return` for functions that have a result
- Functions without a `return` actually return a result `None`

Functions that modify parameters

- Assigning to a parameter is a local change to the local parameter, not the argument
- This leaves `x=10` and `y=11`

```
def f(x):
  x = x+1
  return x

x=10
y=f(x)
```

Functions that modify parameters

- Python passes arguments by assigning their values to the parameters
- But, some values are actually references to "structures": lists, arrays, sequences
- Assigning to the element of a structure does change the value of that element

Functions that modify parameters

- Assigning to an element of a parameter changes that element
- This leaves

```
x=[2,2]
```

```
def f(x):
  x[0] += 1

x=[1,2]
f(x)
```

Functions & modularity

- Functions reduce code duplication
- They also enhance program modularity by allowing complex programs to be broken down into manageable subprograms that can be understood independently
- More to come later in Chapter 9