Functions

Functions

A ***function*** is a subprogram—a small program inside a program—with a name.

- Can be used anywhere anytime by calling its name.
- Most useful when we want to repeat the same basic task (e.g., rounding a real number to an integer).

Why functions?

- Make easier to read and debug.
- Make programs shorter by eliminating repetitive code; if you make a change, you only have to make it in one place.
- Can be used for other purposes in the future…

Encapsulation

- Wrapping a piece of code up in a function is called ***encapsulation***.
- To *encapsulate* means to enclose something in a capsule.

Interface

- When a piece of code is encapsulated in a function, users do not need to know how it is implemented inside.
- Users only need to know the ***interface***: the name, list of parameters and what is being computed (but not how).
- For example, to invoke `math.sqrt(2)`, users do not need to know how to compute square roots.

Parameters and arguments

A variable inside a function whose value is specified by a caller is called a *parameter*.

A value being passed by a caller to a parameter is called an *argument*.

A parameter is always a *variable*.

An argument is always a *value*.

Parameters and arguments

If a function has *n* parameters, a caller must pass *n* arguments.

Arguments are assigned to parameters according to the order alone.

Parameters and arguments

Consider this very simple function.

```
def goo(x, y):
  print '1st value:', x
  print '2nd value:', y
```

What does this do?

This function simply prints the values of the parameters `x` and `y`.

Parameters and arguments

What does this do?

```
>>> def goo(x, y):
      print '1st value:', x
      print '2nd value:', y
>>> goo(3, 4)
1st value: 3
2nd value: 4
```

Parameters and arguments

What does this do?

```
>>> def goo(x, y):
      print '1st value:', x
      print '2nd value:', y
>>> a = 5
>>> b = 6
>>> goo(a, b)
1st value: 5
2nd value: 6
```


Parameters and arguments

What does this do?

```
>>> def goo(x, y):
      print '1st value:', x
      print '2nd value:', y
>>> x = 7
>>> y = 8
>>> goo(y, x)
1st value: 8
2nd value: 7
```


Variables and parameters

Variables and parameters defined inside a function are *local*: they can only be used inside the function.

Parameters and arguments

What does this do?

```
>>> def hoo(x, y):
      n = x
      print '1st value:', n
      print '2nd value:', y
>>> n = 1
>>> y = 2
>>> hoo(y, n)
1st value: 2
2nd value: 1
```


Simple repetition

Loop constructs allow a program to repeat executing sequences of instructions.

The most basic form of loop construct is the *for* statement:

```
for i in range(n):
  <statements>
```

Python repeats `<statements>` `n` times.

Simple repetition

How should we implement this in Python?

```
def n_times(x, n):
  somehow repeat printing x for n times.
```

Here is a solution:

```
def n_times(x, n):
  for i in range(n):
    print x
```

Simple repetition

What does this do?

```
>>> def ioo(n):
      for i in range(n):
        print i
>>> ioo(4)
0
1
2
3
```


Simple repetition

The most basic form of loop construct is the *for* statement:

```
for i in range(n):
  <statements>
```

Python repeats `<statements>` `n` times.

Simple repetition

The most basic form of loop construct is the *for* statement:

```
for i in range(n):
  <statements>
```

As Python repeats `<statements>`, the value of `i` actually grows from `0` through `n-1`.

range()

Let `n` be an integer > `0`.

- `range(n)` is
  `[0, 1, 2, . . ., n-1]`.
- For an integer `m` < `n`, `range(m, n)` is
  `[m, m+1, m+2, . . ., n-1]`.
- For an integer `m` < `n` and an integer `k` > `0`, `range(m, n, k)` is
  `[m, m+k, m+2k, . . ., n-1]`.

range()

Example.

```
>>> print range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> print range(2, 7)
[2, 3, 4, 5, 6]
>>> print range(3, 20, 2)
[3, 5, 7, 9, 11, 13, 15, 17, 19]
```

```
print range(10)       # generates [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print range(3, 10)    # generates [3, 4, 5, 6, 7, 8, 9]
print range(4, 20, 3) # generates [4, 7, 10, 13, 16, 19]

def sum(n):
  "Print the sum of 1 through n."
  sum = 0
  for i in range(n):
    sum = sum + (i + 1)
    print 'When i = ', i, 'sum = ', sum
  print sum

sum(10)

def my_sum():
  "Print the sum of integers in the list [5, 2, 21]."
  sum = 0
  list = [5, 2, 21]
  for i in list:
    sum = sum + i
  print sum

my_sum()

def sum_even(n):
  "Print the sum of even integers between 2 and n."
  sum = 0
  for i in range(2, n + 1, 2):
    sum = sum + i
    print 'When i =', i, 'sum =', sum
  print sum

sum_even(11)

def power(x, n):
  "Print the value of x to the n-th."
  product = 1
  for i in range(n):
    product = product * x
    print 'When i =', i, 'product =', product
  print product

def my_power(x, n):
  "Print the value of x to the n-th."
  print x ** n
```

```
print 'The result of power: '
power(2, 4)

print 'The result of my_power: '
my_power(2, 4)

def letters():
  "Print the letters a through z."
  for i in range(ord('a'), ord('z') + 1):
    print chr(i)

letters()
```