Valued functions

So far all functions we designed output the results of computation via print statements:

```
def foo(<parameters>):
  <statements>
  print ...
```

However, functions can also *return* (and thus represent) values.

Valued functions

So far all functions we designed output the results of computation via print statements:

```
def foo(<parameters>):
  <statements>
  print ...
```


Valued functions

This function `max` *returns* the maximum of two given numbers:

```
def max(x, y):
  if x > y: return x
  else: return y
```

Here, the maximum value will be *returned* to the caller.

Valued functions

This function `max` *returns* the maximum of two given numbers:

```
def max(x, y):
  if x > y: return x
  else: return y
```

Valued functions

When we call a function that returns a value, a function call is an *expression* representing a *value* being returned.

Example.

`max(3, 4)` is an expression representing a value returned by the function `max` with arguments `3` and `4` (that is, `4`).

Valued functions

When we call a function that returns a value, a function call is an *expression* representing a *value* being returned.

Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

print max(3, 4)
```

Output: `4`

Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

print max(3, 4)
```

Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

print max(3, 4) + max(5, 6)
```

Output: 10

Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

print max(3, 4) + max(5, 6)
```


Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

print max(3, max(4, 5))
```

Output: 5

Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

print max(3, max(4, 5))
```

Valued functions

This function `is_even` returns a boolean value: `True` or `False`.

```
def is_even(n):
  if n % 2 == 0: return True
  else: return False
```

Here, `x % y` gives the remainder of `x / y`.

Valued functions

This function `is_even` returns a boolean value: `True` or `False`.

```
def is_even(n):
  if n % 2 == 0: return True
  else: return False
```


Valued functions

What would this output?

```
def is_even(n):
  if n % 2 == 0: return True
  else: return False

print is_even(1823)
```

Output: `False`

Valued functions

What would this output?

```
def is_even(n):
  if n % 2 == 0: return True
  else: return False

print is_even(1823)
```

Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

def is_even(n):
  if n % 2 == 0: return True
  else: return False

print is_even(max(3, 4))
```

Output: `True`

Valued functions

What would this output?

```
def max(x, y):
  if x > y: return x
  else: return y

def is_even(n):
  if n % 2 == 0: return True
  else: return False

print is_even(max(3, 4))
```


Valued functions

This boolean-valued function can be used in an `if` statement. What would this output?

```
def is_even(n):
  if n % 2 == 0: return True
  else: return False

n = input('Enter a number: ')
if is_even(n):
  print 'You entered an even number.'
else:
  print 'You entered an odd number.'
```


Incremental Development

Use incremental development to define a function that computes the average of three numbers.

First try: Syntactically correct and runs.

```
def average(n1, n2, n3):
   return 0.0

print average(1, 2, 4)   # Always returns 0
```

Incremental Development

Third try: Compute the average and display it.

```
def average(n1, n2, n3):
   sum = n1 + n2 + n3
   print 'sum of ',n1,' ',n2,' ',n3,' = ', sum
   average = sum / 3
   print 'average = ' average
   return 0.0

print average(1, 2, 4)    # Is average correct?
```


Incremental Development

Second try: Compute the sum and display it.

```
def average(n1, n2, n3):
   sum = n1 + n2 + n3
   print 'sum of ',n1,' ',n2,' ',n3,' = ', sum
   return 0.0

print average(1, 2, 4)  # Is sum correct?
```


Incremental Development

Fourth try: Fix a bug

```
def average(n1, n2, n3):
   sum = n1 + n2 + n3
   print 'sum of ',n1,' ',n2,' ',n3,' = ', sum
   average = sum / 3.0
   print 'average = ' average
   return 0.0

print average(1, 2, 4)    # Average is correct!
```


Incremental Development

Fifth try: Return the average

```
def average(n1, n2, n3):
   sum = n1 + n2 + n3
   #print 'sum of ',n1,' ',n2,' ',n3,' = ', sum
   average = sum / 3.0
   #print 'average = ' average
   return average

print average(1, 2, 4)    # Average is correct!
```

Incremental Development

- Add code in small increments.
- Run your partial solutions.
- Test and verify before adding more code.
- Use local variables to display partial results.
- Remove ***scaffolding*** and clean up code.
- Make sure your code is readable (understandable).
- Document your code as you go.

Incremental Development

Sixth try: Clean it up

```
def average(n1, n2, n3):
    '''Returns average of its 3 parameters'''
    sum = n1 + n2 + n3
    average = sum / 3.0
    return average

print average(1, 2, 4)    # Average is correct!
```


In Class Exercises

- Incrementally write a valued function that converts fahrenheit to centigrade.
- Incrementally write a valued function that converts kilometers to miles.