


# Functions

CSE 231, Bill Punch


## Functions

- From Mathematics we know that functions perform some operation and return one value.
- They “encapsulate” the performance of some particular operation, so it can be used by others (for example, the `sqrt()` function)

 Michigan State University CSE 231, Fall 2009 Intro to Functions 2


## Characteristics

- Reusable code
  - code in `sqrt()` is reused often
- Encapsulated code
  - implementation of `sqrt()` is hidden
- Can be stored in libraries
  - `sqrt()` is found in the `math` module

 Michigan State University CSE 231, Fall 2009 Intro to Functions 3

## Mathematical Notation


- Consider a function which converts temperatures in Celsius to temperatures in Fahrenheit.
  - Formula:  $F = C * 1.8 + 32.0$
  - Functional notation:  $F = \text{celsius2Fahrenheit}(C)$  where  $\text{celsius2Fahrenheit}(C) = C * 1.8 + 32.0$

 Michigan State University CSE 231, Fall 2009 Intro to Functions 4

## Python Invocation


- Math:  $F = \text{celsius2Fahrenheit}(C)$
- Python, the invocation is much the same  
`F = celsius2Fahrenheit(C)`

Terminology: argument “C”

 Michigan State University CSE 231, Fall 2009 Intro to Functions 5

## Function definition

- Math:  $g(C) = C * 1.8 + 32.0$
- Python  
`def celsius2Fahrenheit (C):`  
`return C*1.8 + 32.0`
- Terminology: parameter “C”

 Michigan State University CSE 231, Fall 2009 Intro to Functions 6

## Terminology

- **Invocation (Call):**  
`F = celsius2Fahrenheit(C)`  
 function name. Must meet regular var naming rules
- **Definition:**  
`def celsius2Fahrenheit(C):`  
`return C*1.8 + 32.0`  
 : indicates the body of the function follows  
 in parenthesis are a list of parameters being passed  
 keyword indicating function being defined  
 indented function body (suite)

Michigan State University  
CSE 231, Fall 2009

Intro to Functions

7

## Operation

`F = celsius2Fahrenheit(C)`

1. Call copies argument C to parameter Temp

2. Control transfers to function "celsius2Fahrenheit"

```
def celsius2Fahrenheit (Temp):
    return temp*1.8 + 32.0;
```

Michigan State University  
CSE 231, Fall 2009

Intro to Functions

8

## Operation (con't)

`F = celsius2Fahrenheit(C)`

3. Expression in celsius2Fahrenheit is evaluated
4. Value of expression is returned to the invoker

```
def celsius2Fahrenheit (Temp):
    return Temp*1.8 + 32.0;
```

Michigan State University  
CSE 231, Fall 2009

Intro to Functions

9

## Verbose Python Invocation

- Math:  $F = g(C)$
- Python  
`TempC=100`  
`TempF = celsius2Fahrenheit(TempC)`  
`print TempC, " in Fahrenheit is: ", TempF`
- TempC is an "argument"

Michigan State University  
CSE 231, Fall 2009

Intro to Functions

10

## Verbose Python Definition

- Math:  $g(C) = C*1.8 + 32.0$
- Python  
`def celsius2Fahrenheit(Temp):`  
`return Temp*1.8 + 32.0`
- Temp is a parameter

Michigan State University  
CSE 231, Fall 2009

Intro to Functions

11

## Matching Argument to Parameter

- The "arguments" of a function invocation list are passed in order to the function parameter list. The names need not match
- The question is, what gets passed?

Michigan State University  
CSE 231, Fall 2009

Intro to Functions

12

## The function namespace

- When a function runs, it defines its own namespace.
- a namespace is simply another matching of names to objects in the system
- Each namespace is its own matching of names to objects, or **local**. The names in the function are only available to the function.



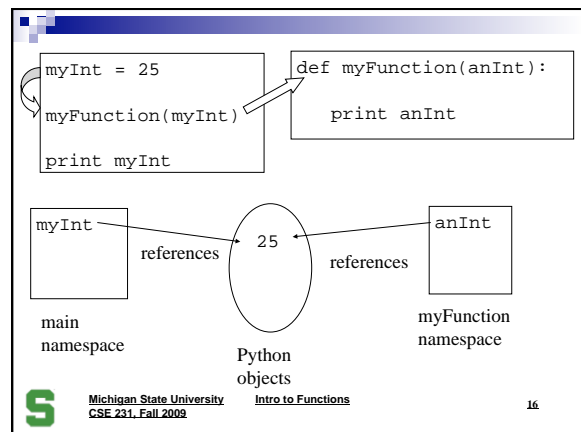
## Passing Arguments

- Python does “pass by reference”
- means that the first argument passes its **namespace reference** to the first parameter, and so on.
- the number of argument/parameter must match
- what gets passed? Well, what does a variable contain?



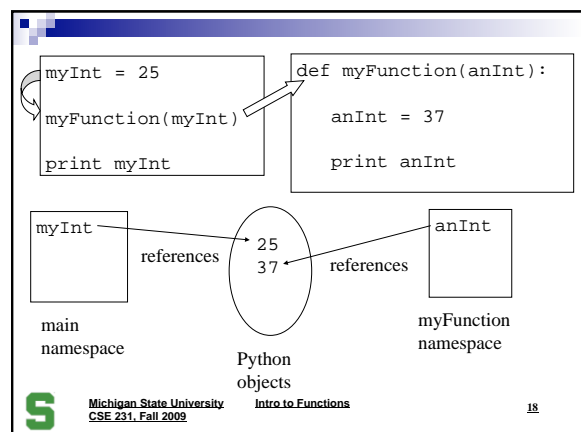
## Pass by reference

- every variable is an entry in a namespace, an association of name and object
- what gets passed is the reference to an object, since that is what a variable has as an association
- Look at an example of passing an integer



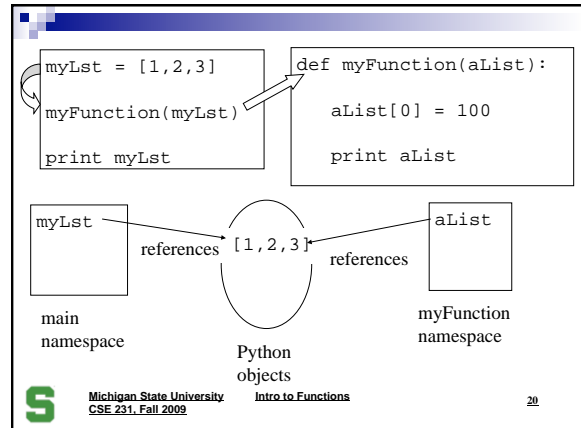
## What got passed

- Both namespaces now refer to the same object, the reference got passed
- However, what happens if the function changes that variable?



## immutables don't change

- the integer is an immutable object, it isn't changed
- the function namespace simply updates the reference to the new object
- the reference in the calling program is unaffected
- But what about mutables?



## mutables change all references

- since the object pointed to is mutable, a change to that object changes it in the calling program as well.
- Thus when both the function and program print, they print:
  - [100, 2, 3]



## Local Objects

- A parameter is a local object which is defined only while the function is executing. Any use of outside the function is an error.
- Any variable assigned in the function is also local and not available outside of the function
- If there is a local var with the same name as another "outside" var, a local var is created with the same name, but:
  - it does not affect the "outside" var
  - the "outside" var value is restored when the function ends
  - different namespaces, get it!!!!



## Example 16 simpleFns

This example illustrates the construction and use of several functions.

## Functions for better design

- Functions are very useful to break the program down into small, understandable, maintainable pieces
- Example:
  - `def get_temp(None)`
  - `def celsius2fahrenheit(temp)`
  - `def display(tempF, tempC)`



## Software engineering

- There is a discipline of computer science dedicated to the systematic development and maintenance of software
- There are a number of approaches that SE use, including: modularization, proveability, testing, refactoring and others



## Refactoring

- Making multiple passes through code to improve its readability and maintainability while not changing (but perhaps improving) its functionality
- Implies that tests are available to apply to code to make sure this is the case
- One refactoring approach is extraction, making complicated code into multiple functions, creating better abstractions



## How to write a function

- Should do one thing. If more than one thing, break into parts. A function *abstracts* one idea
- Should not be overly long (~one page of code). Otherwise break up
- Should be generic in that it could be reused elsewhere in the code
- Should be readable!



## Program Abstraction

- Get Temperature
- Convert Temperature
- Display Temperature



## Needed Behavior

- Please enter a temperature in degrees Celsius: 19.5  
Original: 19.5 C  
Equivalent: 67.1 F



## getTemp

```
def getTemp():  
    t = raw_input("Please enter \  
        temperature in degrees \  
        Celsius: ")  
    return float(t)
```



## Convert

```
def celsius2Fahrenheit(t):  
    result = t*1.8+32.0  
    return result
```



## Display

```
def display(c,f):  
    print "Original: ",c  
    print "Equivalent:",f  
    print
```



## Functions

- Reuse
- Abstraction (Encapsulation)



## PerfectNumbers Fn

## List passed but not returned

- in `classifyValue`, a list is passed but never returned
- This is because the calling program and the function share the same reference, and since the list is mutable a change in the function is a change in the calling program.



## Our own data structure

- The variable `result` defines our own data structure:
  - the first element is a list
  - the second and third are integers
- We add perfect numbers to the first list, and keep the abundant and deficient in the two counts



## isolate complicated printing

- printResults isolates the complicated formatting of the output
- hiding complicated details is one thing that functions do well.



## mileage puzzle

## making a main function

- it is common to create the main function that starts the whole program running
- now when we run our file, it defines the main function which we must call manually.



## extensive use of continue

- checks for 'non-palindromes' under the required circumstances and continues
- makes the process more efficient.



## checking time

```
import time
start = time.time()
... do stuff ...
end = time.time()
print 'It took:',end-start,'seconds'
```



## refactoring

- what if you want to check a different approach to palindrome?
- It is easy to refactor this program. Provide a new function with a different definition to see the effect.
- functions make refactoring easier

