

Objectives

- Announcement: Midterm prep document
- Constants review
- String method review
- Creating your own functions

Oct 8, 2007

Sprenkle - CS111

1

Constants Review

- Constants: don't change for "life" of program
 - During one program execution
 - If it's a constant always (like PI), that's true
- Could play Rainbow Dice to 10, 100, or 1000000
 - Makes easy to change program once at top

Oct 8, 2007

Sprenkle - CS111

2

String Methods

- **Methods**: available operations to perform on strings
 - Slightly different than functions
- Example method: find(substring)
 - Finds the index where substring is in string
 - Returns -1 if substring isn't found
- To call a method:
 - `<string>.methodname([arguments])`
 - Example: `filename.find(".py")`

Oct 8, 2007

Executed on this string

Sprenkle - CS111

3

Common String Methods

Method	Operation
<code>center(width)</code>	Returns a copy of string centered within the given number of columns
<code>count(sub[, start [, end]])</code>	Return # of non-overlapping occurrences of substring <code>sub</code> in the string.
<code>endswith(sub), startswith(sub)</code>	Return <code>True</code> iff string ends with/begins with <code>sub</code>
<code>find(sub[, start [, end]])</code>	Return first index where substring <code>sub</code> is found
<code>isalpha(), isdigit(), isspace()</code>	Returns <code>True</code> iff string contains letters/digits/whitespace only
<code>lower(), upper()</code>	Return a copy of string converted to lowercase/uppercase

Oct 8, 2007

Sprenkle - CS111

`string_methods.py`

4

String Methods vs. Functions

- Functions: all "input" as arguments/parameters
 - Example: `len` is a built-in function
 - Called as `len(str)`
- Methods: "input" are argument/parameters **and** the string the method was called on
 - Example call: `str.upper()`

Oct 8, 2007

Sprenkle - CS111

5

Practice Using String Methods

- Modify `binaryToDecimal.py` to verify that the entered string contains only numbers
 - Use one of the string methods
 - How could we make sure that it contains only 0s and 1s?

Oct 8, 2007

Sprenkle - CS111

6

Functions

- We've used several built-in functions
 - len, input, raw_input
 - Functions from modules, e.g., math and random
- **Functions** are small pieces of code that can be used in other pieces of code
 - Subprograms that have been given a name
 - Have 0 or more inputs
 - Produce 0 or 1 outputs
- Define our own functions!

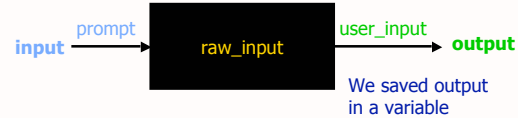
Oct 8, 2007

Sprenkle - CS111

7

Functions

- Function is a **black box**
 - Implementation doesn't matter
 - Only care that function generates appropriate output, given appropriate input
- Example:
 - Didn't care how **raw_input** function was implemented



Oct 8, 2007

Sprenkle - CS111

8

Why write functions?

- Allows you to break up a hard problem into smaller, more manageable parts
- Makes your code easier to understand
- Hides implementation details (*abstraction*)
 - Provides interface (input, output)
- Makes part of the code reusable so that you:
 - Only have to type code once
 - Can debug it all at once
 - Isolates errors
 - Can make changes in one function (maintainability)

Oct 8, 2007

Sprenkle - CS111

9

Comparison of Code Using Functions

- Without functions:
 - menu_withoutfunc.py
- With functions
 - menu_withfunctions.py

Oct 8, 2007

Sprenkle - CS111

10

Example Program

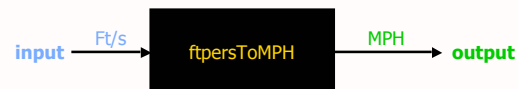
- Lab 2, Problem 1
 - Any place to make a function?
 - Any place that has some useful code that we may want to reuse?

Oct 8, 2007

Sprenkle - CS111

11

Convert ft/s to mph



- Input: ft/s
- Output: mph

Oct 8, 2007

Sprenkle - CS111

12

Syntax of Function Definition

Diagram illustrating the syntax of a function definition:

```
def ftpsToMPH(ftp) :
    SECOND_TO_HOUR = 3600
    FEET_TO_MILE = (1.0/5280)
    result = ftp * SECOND_TO_HOUR * FEET_TO_MILE
    return result
```

Labels in the diagram:

- Keyword:** `def`
- Function Name:** `ftpsToMPH`
- Input Name/Parameter:** `(ftp)`
- Function header:** `def ftpsToMPH(ftp) :`
- Body (or function definition):** The lines of code inside the function.
- Keyword:** `return` (How to give output)

Oct 8, 2007

Sprengle - CS111

13

Where are functions in the code?

- Can be defined in script before use (calling it)
- Could be in separate **module**
 - Import to use in script
 - Example: `menu.py`
 - Define in modules when functions are reusable for many different programs
 - Benefits: shorter code (no function defs), isolate testing of function, write "test driver" scripts to test functions separately from use in script

Oct 8, 2007

Sprengle - CS111

14

Parameters

- The inputs to a function are called **parameters** or **arguments**
- When **calling**/using functions, parameters must appear in same order as in the function header
 - Example: `round(x, n)`
 - **x** is float to round
 - **n** is integer of decimal places to round to

Oct 8, 2007

Sprengle - CS111

15

Parameters

- **Formal Parameters** are the variables named in the the function definition.
 - **Actual Parameters** are the variables or literals that really get used when the function is called.
- Diagram illustrating the difference between Formal and Actual parameters:
- ```
def round(x, n) :
 roundCelc = round(celc, 2)
```
- Labels in the diagram:
- Formal:** `x` and `n` in the function definition.
  - Actual:** `celc` and `2` in the function call.

Formal & actual parameters must match in order, number, and type!

Oct 8, 2007

Sprengle - CS111

16

## Function Output

- When the code reaches a statement like **return x**
  - x** is the output returned to the place where function called and the function stops
  - For functions that don't have explicit output, return does not have a value with it
    - **return**
    - Optional: don't need to have return (see `menu.py`)

Oct 8, 2007

Sprengle - CS111

17

## Calling your own functions

```
jennifer_mph = ftpsToMPH(jen_ft_sec)
laura_mph = ftpsToMPH(laura_ft_sec)
venus_mph = ftpsToMPH(venus_ft_sec)
```

Diagram illustrating the components of a function call:

- Output is assigned to:** `venus_mph`
- Function Name:** `ftpsToMPH`
- Input:** `venus_ft_sec`

Oct 8, 2007

Sprengle - CS111

18

## Flow of Control

- When you call the function, the computer jumps to the function and executes it
- When it is done, it returns to the same place in the first code where it left off

```
COURT_LEN = 78
#Make calculations to ft/s
j_ft_s = COURT_LEN / j_Speed
j_mi_hr = ftpersToMPH(j_ft_s)

def ftpsToMPH(ftps):
 S2HR = 3600
 FT2MI = (1.0/5280)
 result = ftps * S2HR * FT2MI
 return result
```

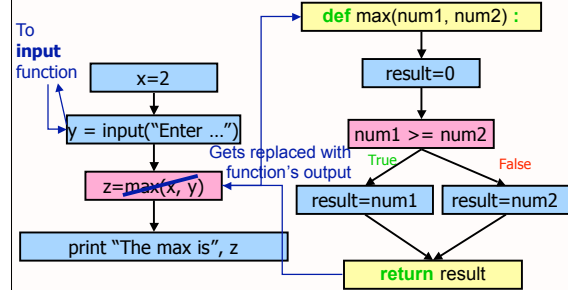
Note: not great variable names

Oct 8, 2007

Sprengle - CS111

19

## Flow of Control



Oct 8, 2007

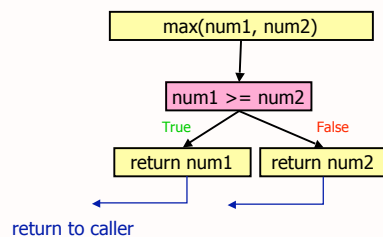
Sprengle - CS111

20

## Flow of Control: Using return

```
def max(num1, num2):
 if num1 >= num2:
 return num1
 else:
 return num2
```

```
x=2
y=6
z = max(x, y)
```



Oct 8, 2007

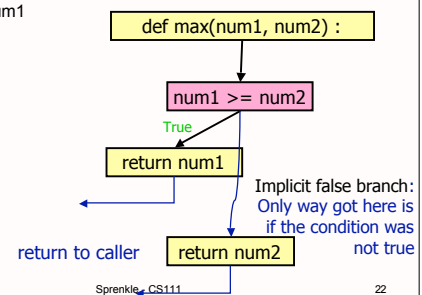
Sprengle - CS111

21

## Flow of Control: Using return

```
def max(num1, num2):
 if num1 >= num2:
 return num1
 return num2
```

```
x=2
y=6
z = max(x, y)
```



Oct 8, 2007

Sprengle - CS111

22

## Passing Parameters

- Only **copies** of the actual parameters are given to the function
- The actual parameters in the calling code do not change.
- Swap example:
  - Swap two values in script
  - Then, put into a function

Oct 8, 2007

Sprengle - CS111

23