Objectives

- Review: string format
- Functions
- Import
- Intro to design patterns
- Definite loops

Review: Formatting

- What data type does string formatting give you?
  - For example, what data type would `"%6.2f" % expense` give back?
- What is the format specifier's *code* for ints? Floats? Strings?
- What is the format specifier for right-justifying a float within 10 spaces that displays 3 decimals?

Example: Printing Out Tables

- A table of temperature conversions

```
Temp F      Temp C      Temp K
------      ------      ------
-459.7      -273.1         0.0
   0.0       -17.8       255.2
  32.0         0.0       273.1
```

- If we want to print data in rows, what is the template for what a row looks like?
  - How do we make the column labels line up?

Parts of an Algorithm

- Input, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - **Call**, reuse similar techniques

FUNCTIONS

Built-in Functions

- Functions perform some task
  - May take **arguments/parameters**
  - May **return** a value that can be used in assignment

What does it do?
How does it do it?

We don't know ***how*** it does it, but it's okay because it doesn't matter →as long as it **works**!

Built-in Functions

Argument/parameter list (input)

- Syntax:
  - `func_name(arg0, arg1, …, argn)`
- Depending on the function, arguments may or may not be required
  - [ ] indicate an optional argument
- Semantics: depend on the function

Example Built-in Functions

Known as function's ***signature***

Template for how to "**call**" function

Optional argument

- `raw_input([prompt])`
  - If prompt is given as an argument, prints the prompt without a newline/carriage return
  - If no prompt, just waits for user's input
  - **Returns** user's input (up to "enter") as a **string**
- `input([prompt])`
  - Similar to `raw_input` but **returns** a **number**

More Examples of Built-in Functions

Function Signature	Description
`round(x[,n])`	Return the `float` **x** rounded to **n** digits after the decimal point If no **n**, round to nearest `int`
`abs(x)`	Returns the absolute value of **x**
`type(x)`	Return the type of **x**
`pow(x, y)`	Returns x^y

Terminal

Using Functions

- Example use: Alternative to exponentiation
  - Objective: compute -3^2
  - Python alternatives:
    - **pow**(-3, 2)
    - (-3) ** 2
- We often use functions in assignment statements
  - Function does something
  - Save the output of function in a variable

```
roundx = round(x)
```


Python Libraries

- Beyond built-in functions, Python has a rich **library** of functions and definitions available
  - The library is broken into **modules**
  - A ***module*** is a file containing Python definitions and statements
- Example modules
  - `math` — math functions
  - `os` — OS functions
  - `network` — networking functions

Math Module

- Defines constants (variables) for `pi` (i.e., π) and `e`
  - These values never change, i.e., are ***constants***
  - Recall: ***we*** name constants with all caps
- Defines functions such as

Function	What it Does
`ceil(x)`	Return the ceiling of x as a float
`exp(x)`	Return e raised to the power of x
`sqrt(x)`	Return the square root of x

Using Python Libraries

- To use the definitions in a module, you must first **import** the module
  - Example: to use the **math** module's definitions, use the the import statement: **import math**
  - Typically import statements are at ***top*** of program
- To find out what a module contains, use the **help** function
  - Example within Python interpreter

```
import math
help(math)
```


Using Definitions from Modules

- Prepend constant or function with "**modulename**."
  - Examples for constants:
    - `math.pi`
    - `math.e`
  - Examples for functions:
    - `math.sqrt`
- Practice
  - How would we write the expression $e^{i\pi} + 1$ in Python?

Alternative Import Statements

```
from <module> import <defn_name>
```

- Examples:
  - `from math import pi`
    - Means "import pi from the math module"
  - `from math import *`
    - Means "import *everything* from the math module"
- With this **import** statement, don't need to prepend module name before using functions
  - Example: `e**(1j*pi) + 1`

Benefits of Using Python Libraries/Modules

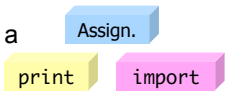
- Don't need to rewrite someone else's code
- If it's in a module, it is very *efficient* (in terms of computation speed and memory usage)

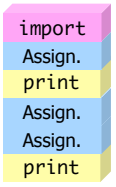
Finding Modules To Use

- How do I know if functionality that I want already exists?
  - Python Library Reference: `http://docs.python.org/lib/lib.html`
- Example: `string` module has functions for manipulating strings
- For the most part, in the beginning you will write most of your code from scratch

Programming Building Blocks

- Each type of statement is a building block
  - Initialization/Assignment
    - Arithmetic, string concatenation, functions
  - Print
  - Import
- We can combine them to create more complex programs
  - Solutions to problems





Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution

Design Patterns

- General, repeatable solution to a commonly occurring problem in software design
  - Template for solution
- Example (Standard Algorithm)
  - Get input from user
  - Do some computation
  - Display output

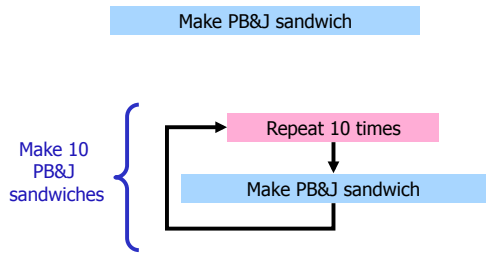
Assign.	`x = input("…")`
Assign.	`ans = …`
`print`	`print ans`

FOR LOOPS

Parts of an Algorithm

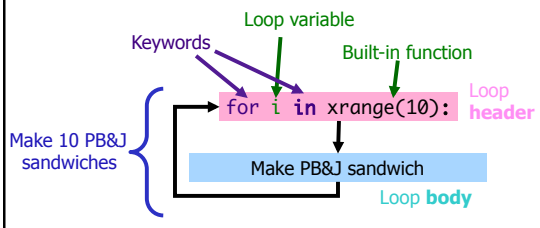
- Input, Output
- Primitive operations
  - What data you have, what you can do to the data
- Naming
  - Identify things we're using
- Sequence of operations
- Conditionals
  - Handle special cases
- Repetition/Loops
- Subroutines
  - Call, reuse similar techniques

Looping/Repetition



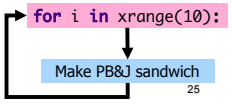
The `for` Loop

- Use when know how many times loop will execute
  - Repeat N times



What Goes in the Loop Body?

- Make PB&J Sandwich
  1. Gather materials (bread, PB, J, knives, plate)
  2. Open bread
  3. Put 2 pieces of bread on plate
  4. Spread PB on one side of one slice
  5. Spread Jelly on one side of other slice
  6. Place PB-side facedown on Jelly-side of bread
  7. Close bread
  8. Clean knife
  9. Put away materials



What Goes in the Loop Body?

- Make PB&J Sandwich
  1. Gather materials (bread, PB, J, knives, plate)
  2. Open bread

  Initialization

  Loop Body
  3. Put 2 pieces of bread on plate
  4. Spread PB on one side of one slice
  5. Spread Jelly on one side of other slice
  6. Place PB-side facedown on Jelly-side of bread
  7. Close bread
  8. Clean knife
  9. Put away materials

  Finalization

Using the `for` Loop

- Use when know how many times loop will execute
  - Repeat N times

Times to repeat

```
for i in xrange(10):
    statement_1

    statement_2
    …
    statement_n
```

"Body" of `for` loop
- Gets repeated
- Note indentation

Using the `for` Loop

- If only ***one*** statement to repeat

```
for variable in xrange(5): print "Hello!"
```


Analyzing `xrange()`

- **`xrange`** is a built-in function
- What does **`xrange`** do, exactly, with respect to the loop variable `i`?

```
for i in xrange(10):
   squared = i * i
   print i , "^2 =\t", squared

print i
```


`xrange([start,] stop[, step])`

- What does the above signature mean?

xrange([start,] stop[, step])

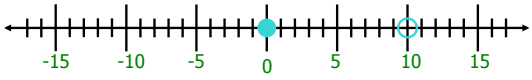
- 1 argument: xrange(stop)
- 2 arguments: xrange(start, stop)
- 3 arguments: xrange(start, stop, step)

xrange([start,] stop[, step])

- 1 argument: xrange(stop)
  - Defaults: start = 0, step = 1
  - Iterates from 0 to stop-1 with step size=1
- 2 arguments: xrange(start, stop)
  - Default: step = 1
  - Iterates from start to stop-1 with step size=1
- 3 arguments: xrange(start, stop, step)
  - Iterates from start to stop-1 with step size=step

xrange()

- xrange is a built-in **function**
  - 1 argument: xrange(stop)
  - 2 arguments: xrange(start, stop)
  - 3 arguments: xrange(start, stop, step)

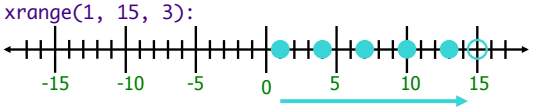


[start, stop)

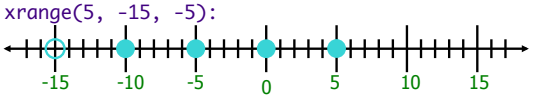
```
xrange(10)
xrange(0,10)
xrange(0,10,1)
```


xrange()

xrange(1, 15, 3):



xrange(5, -15, -5):

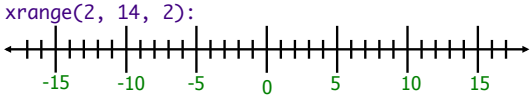


Practice

Place these: ● ○

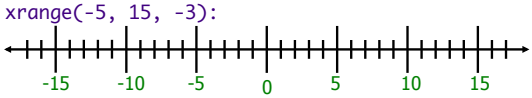
Which direction?

xrange(2, 14, 2):



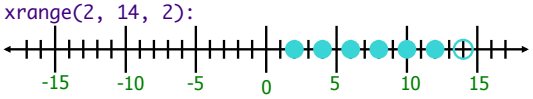
xrange(8, -10, -3):

xrange(-5, 15, -3):

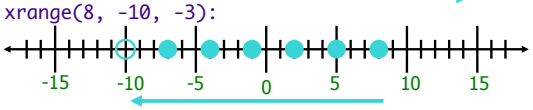


Practice Solution

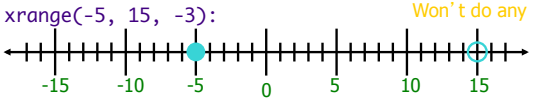
xrange(2, 14, 2):



xrange(8, -10, -3):



xrange(-5, 15, -3): Won't do any



Programming Practice

- Add 5 numbers, inputted by the user
  - After implementing, simulate running on computer

Accumulator Design Pattern

1. Initialize accumulator variable
2. Loop until done
   - Update the value of the accumulator
3. Display result

Programming Practice

- Average 5 numbers inputted by the user

This Week

- Tuesday: Lab 2
  - Lab due on Friday
- Wednesday: Advanced for Loop
- Friday Broader Issue: IBM's Watson