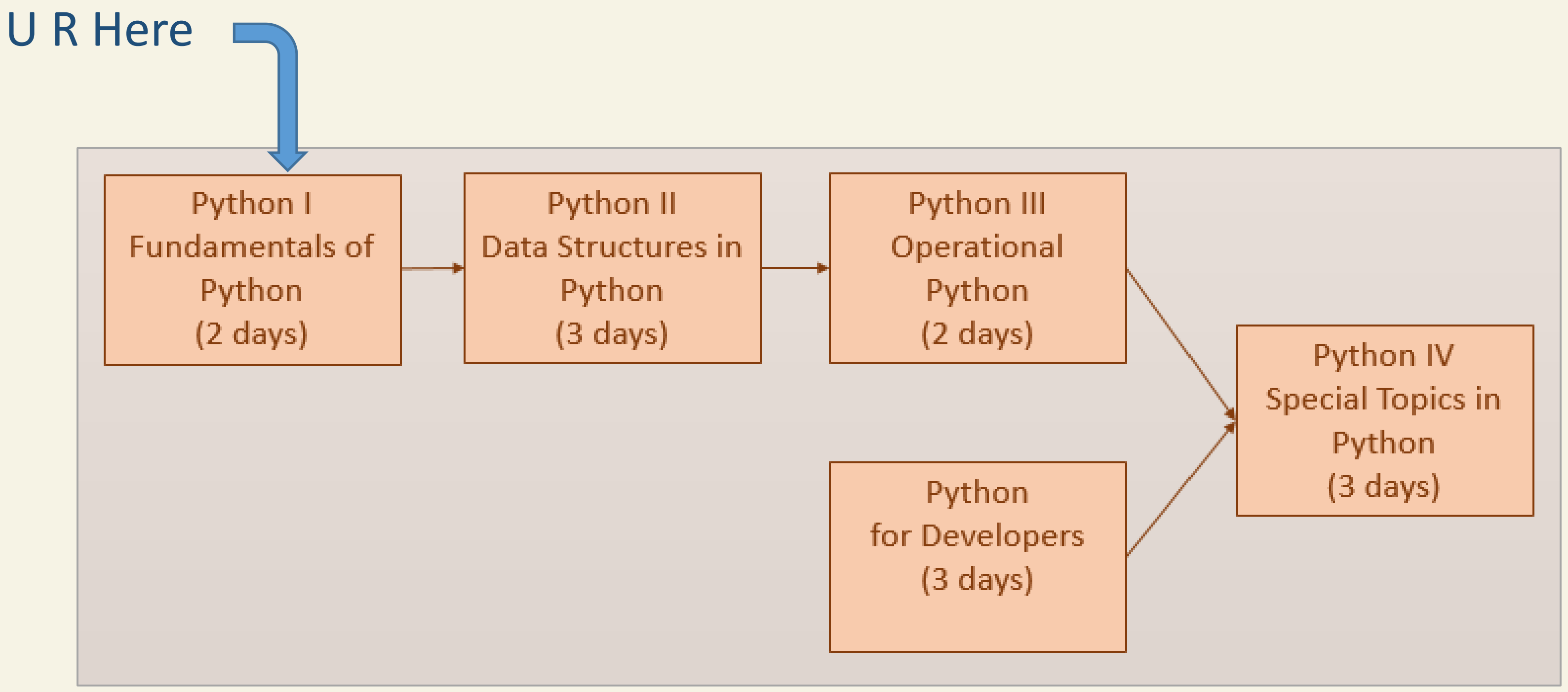
Python I
Welcome to Programming

Rev:09/26/2016

Python Classes



Objective:

- 20% Lecture
- 80% Lab

Introductions

- Please mention:
  - First name
  - What you do at Rackspace
  - Any programming or scripting experience you may have.
  - What you hope to get out of the class.

PAPERWORK

- NLC registration
- RU roster
- Completed labs will be made available daily through the cloud.
- If you have questions, my e-mail is:
  - weastridge@gmail.com
  - Make the subject: Python

Introduction to Programming

- Objective - Learn the basics of programming using the Python language.
  - Has its own set of rules.
  - Basic principles are the same as any other language.
- Recognize that a programming language is actually a language to be mastered.
  - As with any language, it requires constant use and study to attain fluency and fluency is the goal.
  - Taking a class is only a start.

Castle Lab Environment

- ID – student, password - student
- IDLE Install
  - For CentOS: sudo yum install python-tools
  - For Ubuntu et al: sudo apt-get install idle
  - Windows and OS X come with IDLE installed
    - Macs with retina displays may require a larger font size than the default to make underscores show properly.
- CentOS7 at the Castle has IDLE already installed.
- To create an icon for IDLE:
  Drag the file Idle.desktop to the desktop, double-click it and trust it. This is your icon for IDLE.
- Data
  - go to tinyurl.com/py1dataz
  - This file contains documents, samples and data for labs

Agenda

Background. General information

Define / Set up lab environment

Basics of sequence, selection and iteration

Functions and code reuse

Intercepting errors

Reading and writing files

Resources

- Two books in PDF format oriented to Python 2.7:
  - Python for Informatics – this is a practical book for beginners, but in chapter 11 he transitions to some pretty advanced material.
  - Think Python – a larger somewhat more comprehensive book with more difficult exercises.
- Python Notes – collections of useful information in one place.
- A copy of the slides used in this class.
- A set of screen shots showing samples of basic Python code.
- A set of completed Labs will be provided at the end of each day.

Programming Languages

- Machine code
- Assembly Language
- Higher-Level Languages
  - Fortran and COBOL (1950’s)
  - C (1969) – Strictly imperative
  - C++ (1979) – Originally called C with Classes – Object Oriented
  - Java (1995) – Currently one of the most popular languages
  - Python (1994 – 1.0) - Started as a hobby by Guido van Rossum in 1989
    - Universities find students learn Python much more quickly than Java
    - Rackspace does a great deal of development in Python

Language Comparisons – Hello World

C Language

```
#include<stdio.h>
main()
{
    printf("Hello World!");
}
```

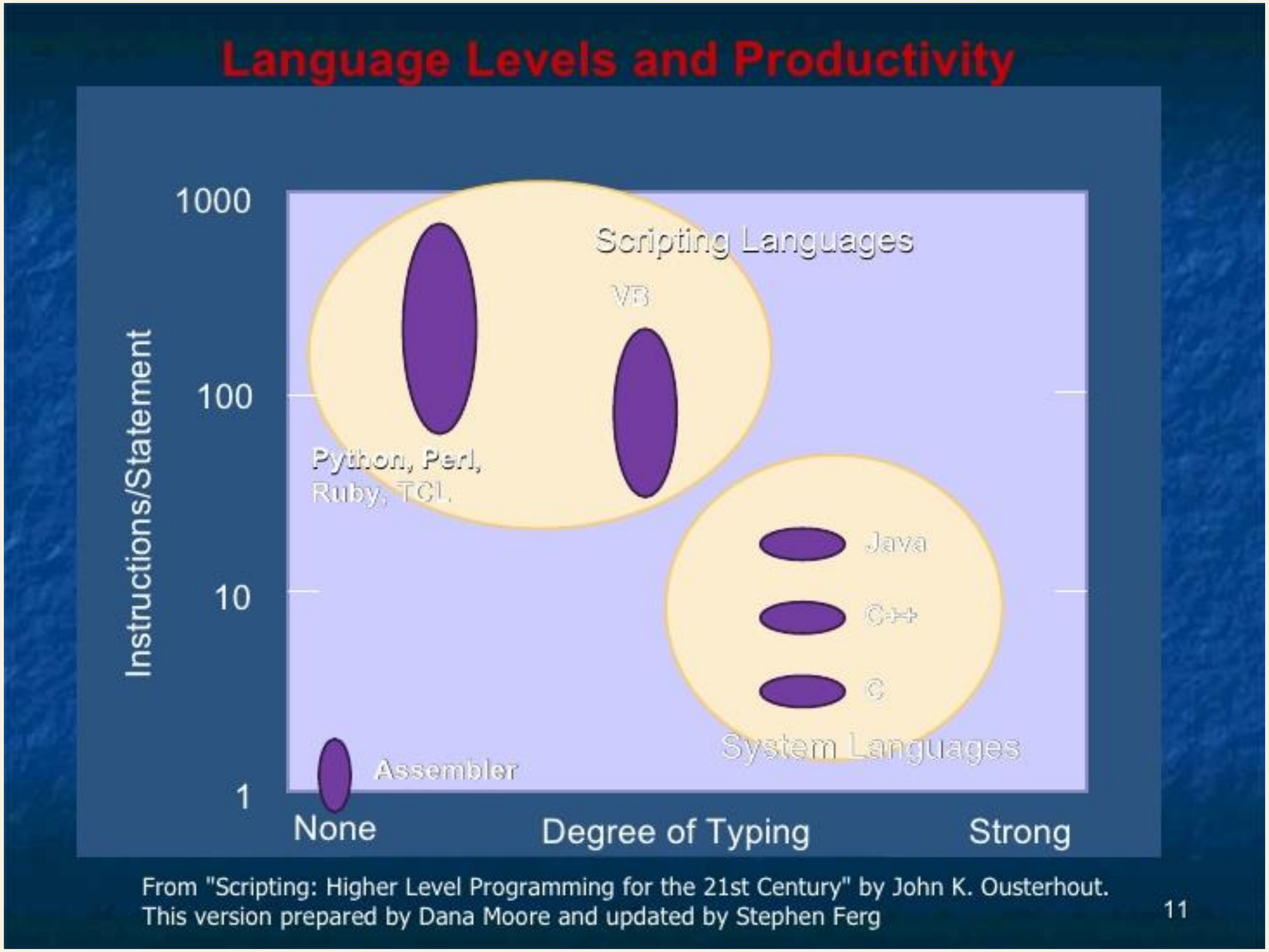
Java Language

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Python – Hello World

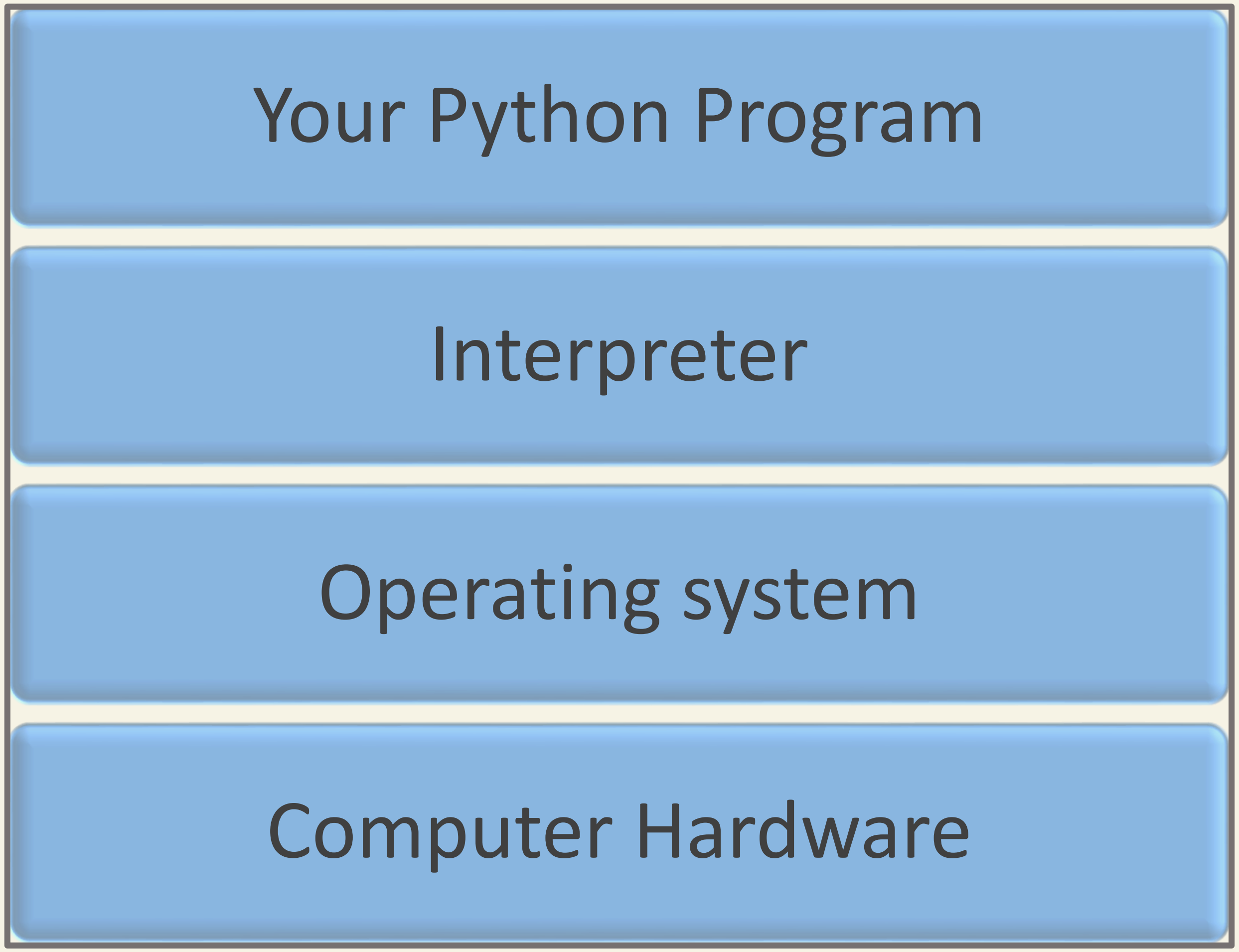
```
print "Hello World!"
```

So, why Python?



From "Scripting: Higher Level Programming for the 21st Century" by John K. Ousterhout. This version prepared by Dana Moore and updated by Stephen Ferg

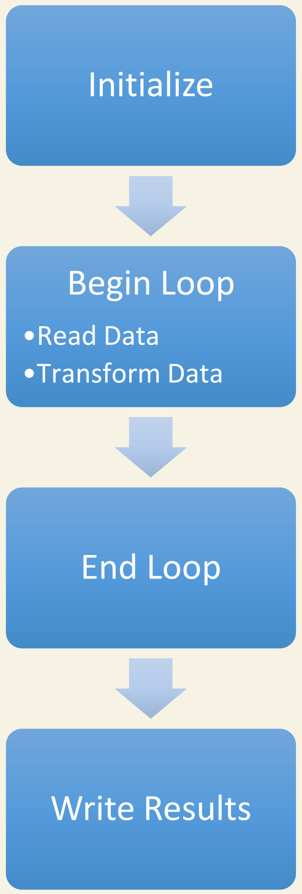
THE NOTIONAL MACHINE



From Concept to Code

- Create a plan.
- Determine how to go from your plan to actual code.
- Always have a working program.
- Resolve problems as you encounter them.
  - Review the reference material you get in this class.
  - The web is loaded with information (much of it confusing to beginners).
  - Be sure to do everything you can before asking your team members.
  - ASK AS MANY QUESTIONS AS YOU WANT IN CLASS.

A Simple Program



Sequence

Selection

Iteration

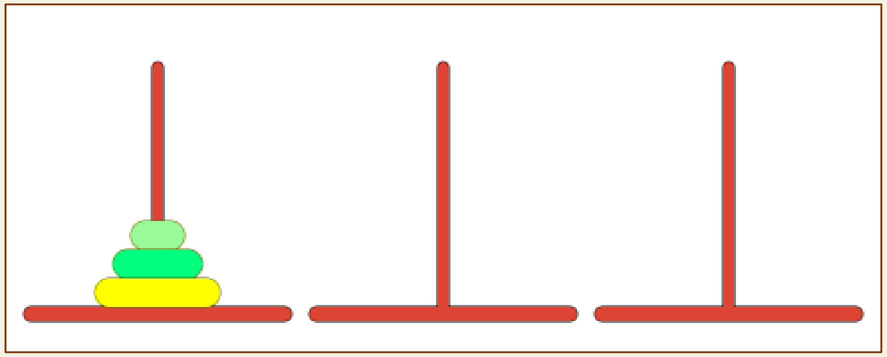
Sequence Exercise

The Towers of Hanoi is a good exercise in sequence. The objective is to move the three disks from the left tower to the right tower. The rules are that you can move only one disk at a time. Also, at no time can a larger disk be placed on top of a smaller disk. You can practice this exercise at this site. You are to tell the computer how to accomplish this task Create a sequence of commands using the options below to tell the computer exactly how to achieve the above goal.

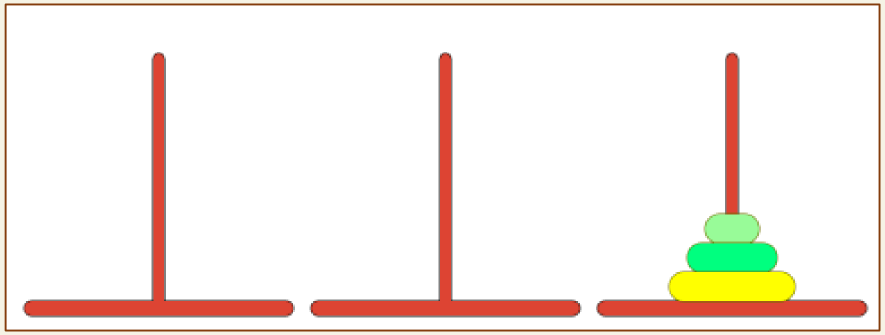
Move the (small/medium/large) disk to the (right/center/left) tower

Inside the parentheses, select one item for each command you write. A minimum of seven commands is required.

Starting positions



Ending positions



Planning

- Planning is essential to writing a program.
- Always understand how you are going to approach a problem.
- For example:
    - What calculations will you have to perform?
    - What data will you need?
    - Where will it come from?
    - How will you preserve/save it?
    - What will you do with the results?
- These are just a few of the questions you need to answer.

ON BOARD LAB

Plan a program that averages some numbers entered from the keyboard. When the operator enters a 'Q', they are finished entering numbers. Solve this problem using ordinary English.

Semantics

Bit – Binary digit - the smallest unit of computer storage.

Byte – Grouping of 8 bits. Can store a character (e.g., letter a)

Binary – Numbering system used by computers to do math.

- Example – decimal number 147 in binary is 10010011
- Numbers used in math usually occupy multiple bytes.

ASCII – A system for representing human-readable characters in a computer. See chart in Python Notes

Our Lab Environment

- Python 2.7 (aka CPython)
- IDLE
  - Rudimentary IDE
  - Really primarily an editor/executor
  - Shell operations vs program creation windows
- Command line / Vim
- Do "Hello world" both ways

LAB

LPTHW Lessons 2 and 4

http://learnpythonthehardway.org/book/

PRINTING

- The print statement versus the print function
- The print statement is used to output one or more items. These can be literals, variables or expressions
- Can format the output. We will do this later.
- Automatically skips to next line when done, unless ended with ',‘

Identifiers & Type

- Pieces of memory have names you give them (usually called identifiers or variables)
- The identifier (or variable) points to (binds) the memory location where the data is.
- The data has a type and a value

x = 10   x is an integer (int)

y = 13.7 y is a decimal number (float)

z = 'Some text' z is a string (str)

Identifier Naming Requirements

- Must start with a letter or underscore
- Are case sensitive
- Can only have letters, numbers, and underscores
- Can be any length, so use a descriptive name
- Cannot be a reserved word. See Python Notes for a list of reserved words.

Literals

- Literals are sequences of characters, such as a number or text.
- Literals represent themselves literally – they are not identifiers pointing elsewhere
- Literals are very handy for initializing variables
- Ex.:

  ```
  x = 10
  y = 12.3
  z = ‘This is text’
  ```
- In Python, the only way to define a variable is to initialize it.

Math Operators

**	Exponentiation
/	Division (works differently in Python 3)
*	Multiplication
+	Addition
-	Subtraction
%	Modulus (Remainder)
//	Integer/Floor Division

Assignment

- Assignment makes a variable on the left side of the "=" sign (equal) to an evaluated expression on the right side.

  - x = 12.3
  - x = x + 1 or
  - x += 1 (shorthand)
  - y = x + 117
  - z = 12 * x / y

  - a = "This is a line of text"
  - b = " and so is this"
  - c = a + b
  - d = 20 * '-'

  There can be more than one variable on the left side.

  - x, y = y, x

  Sample – a0Assignments.jpg

Basic Data Types

integer

float

string

boolean

None

LAB

LPTHW Lessons 7, 9 and10

- If you try the “tiny piece of fun code to try out” in lesson 10, be sure to do it in command line, not idle. Issue cntrl>c to get out of the loop.

http://learnpythonthehardway.org/book/

Expressions

- An expression is a combination of one or more identifiers, operators, literals, and/or “functions”
- Expressions compute something
- Expressions can occur on the righthand side of the '=', as arguments, and many other places.
- Boolean expressions are either True or False.

Math Operators

**	Exponentiation
/,//,%	Division
*	Multiplication
+	Addition
-	Subtraction

Order of Execution (Precedence)

- Exponentiation
- Multiplication/Division (left to right)
- Addition/Subtraction (left to right)

Only parentheses change the order of execution.

Operators - Do Each of These in the Shell

```
a = 7 / 3
a = 7.0 / 3
a = 7.0 // 3
a = 7 // 3
a = 7 % 3
x = 12.3
y = 10 * x
z =  y – 14 / 2
z =  (y – 14) / 2
```

Lab01 - Formulas

- Create a program to solve these problems and print the results. Place each value specified by the problem into a variable before doing any calculations. Do not worry about the formatting of the answers. Your program should be such that you can change any of the figures below and get a new answer.
  - You bought 125 shares of a stock at $25.32 and you sold it at $48.97. What is the profit?
  - A product with a price of $127.99 is going on sale. If the price is reduced by 16%, what will the new price be?

Syntax, Semantics and Failure

- You may have experienced some of these already
- Syntax – basic structure – commands, punctuation
- Semantics – meaning
- Failure – your program fails when you run it. You have to find out why.

Formatting – Newer Style

- In the Introductory classes, we previously used the old style formatting.
- Learn Python the Hard Way and many other sites still use the style.
- You will see both the older and newer styles used in Stack Overflow as well.
- It is important to recognize and know both. They are both covered in Python Notes. Some completed labs show how to use both.
- The next several slides and the labs in the rest of this class use the newer style.

Formatting Data into Strings

General example:

‘insert text here with {0} {1}'.format(variable, “literal string”)

Abbreviated general format of a formatting sequence:
{[seq#] ":" [width] [","] ["." prec] [type]} - See Python Notes for more detail

- The sequence number [seq#] is optional. If missing, variables/literals are formatted in the order given (python 2.7+).
- Width is used to expand a formatted item beyond the default. In the expanded width, numbers are right justified, text left justified.
- The ',' is used as a thousands separator in larger numbers.
- The ".prec" specifies the number of decimal places to display.
- The short list of valid types are s, d and f .
    - s – strings, d – integers, f – floating-point numbers

Formatting Data into Strings

Abbreviated general format: {[seq#] ":" [width] [","] ["." prec] [type]}

Examples:

a = 12

b = 17.426

x = 'Some text {} more text {}'.format(a, b)

Result stored as x – 'Some text 12 more text 17.426'

x = 'Some text {0} more text {1}'.format(a, b)

Result stored as x – 'Some text 12 more text 17.426 '

x = 'Some text {1} more text {0}'.format(a, b)

Result stored as x – 'Some text 17.426 more text 12 '

x = 'Some text {0} more text {1:.2f}'.format(a, b)

Result stored as x – 'Some text 12 more text 17.43' (Note rounding)

See examples in Sample folder - a2Formats.jpg and a3Formats.jpg

Formatting Data into Strings

Abbreviated general format: {[seq#] ":" [width] [","] ["." prec] [type]}

General examples:

a = 1234567.889

x = 'I would like to have ${}'.format(a)

Result stored as x – 'I would like to have $1234567.889'

x = 'I would like to have ${0:,.2f}'.format(a)

Result stored as x – 'I would like to have $1,234,567.89 ' (Add separators – note rounding)

x = 'I would like to have ${0:15,.2f}'.format(a)

Result stored as x – 'I would like to have $ 1,234,567.89'

When the width specified is larger than necessary to accommodate the result, numbers are right justified and everything else is left justified.

Lab02 - Formulas

Redo the last lab formatting the output.

Getting Keyboard Input

- raw_input() function
- strings vs. floats and integers
- float() and int() functions
- See bRaw_input-Float-Int.jpg in Samples

PYTHON HELP

- Google!
- Interactive shell - help()
- At bash prompt: pydoc
  - e.g. pydoc -k string or pydoc module
- Windows Powershell
  - python –m pydoc --------
- Pydoc documentation
  - pydoc pydoc

LAB 03

- LPTHW 11-12. Be sure not to skip the Pydoc exercise.
  http://learnpythonthehardway.org/book/

- Plan a program that reads a temperature in degrees Fahrenheit from the keyboard, then prints the equivalent Centigrade temp.

- Write the source code to implement your plan.

- Remember:
  Fahrenheit = 9/5*Centigrade + 32
  Centigrade = 5/9*(Fahrenheit – 32)

Note – Make sure you keep all copies of your programs from now on. You will be re-using this code often.

Python Indentation

- The way Python uses indentation is unique
- Indentation shows where blocks/suites of code begin and end.
- The standard is to use 4 spaces per indentation level (not tabs).
- IDLE will do this for you automatically
    - If it gets confused, it's probably because of something you have done.
    - It replaces tabs with 4 spaces. Multi-line indent/unindent capability.
    - gedit can be set to replace tabs with 4 spaces also.
    - Notepad++ works well in Windows and inserts spaces in place of tabs.

If … (Single Branch Selection)

OXYGEN
OXIGENO
OXYGENE
1
2
3
4

The instructions given consider situations other than the basic one.

"if you are travelling with a child, secure your mask first, then secure the mask of your child.

Putting on an airline oxygen mask

1. secure your mask
2. if you are travelling with a child then
3. secure the mask of the child
4. sit back

Sequence– with child

1->2->3->4

Sequence – without child

1->2->4

What Should I Do?

Reading a book

```
If I have an unread book on my bookshelf:
    Pick a book that you have not read.
    Read the book.
    Put the book back
Otherwise:
    Go out and buy a new book.
    Read it.
    Put it on the bookshelf.
```

One of these sets of instructions will be followed but not both.

Comparison Operators

- `==` Equal
- `!=` Not Equal
- `>` Greater Than
- `<` Less Than
- `>=` Greater Than or Equal
- `<=` Less Than or Equal
- in, not in, is, not is
- not, and, or

Making Decisions

- All languages make decisions, usually if statement.
- if or if/else.
- Each option (if, else) is followed by a suite of statements
    - At most, only one suite is executed
- For a simple if statement, the suite of statements following it are executed if the comparison is true. Otherwise, they are skipped.
- For if/else, one (and only one) suite will be executed. In any case, something will be excuted.
- Demo with sample : (c1If-Else.jpg)

Lab 04

Create a new program from the previous lab to print out a description along with the centigrade temperature. For Fahrenheit temperatures that are above 90 degrees, print “It’s hot outside”. Otherwise, print “It’s not hot outside”. This printing should follow the printing showing the conversion from Fahrenheit to Centigrade.

What Should I Do?

Reading a book

```
If I have an unread book on my bookshelf:
    Pick a book that you have not read.
    Read the book.
    Put the book back
Else if I have enough money:
    Go out and buy a new book.
    Read it.
    Put it on the bookshelf.
Else:
    Get a book from the library.
    Return it before you incur fines.
```

Only one of these sets of instructions will be followed.

Making Decisions

- The if statement has an expanded option for decision making.
- if or if/else or if/elif/else. (Use elif for more than two options)
- Each option (if, elif, else) is followed by a suite of statements
  - At most, only one suite is executed
- There can be multiple elif statements.
- Regardless of the number of elif statements, one (and only one) suite will be executed.
- Demo with samples. : (c2If-Else.jpg)

Lab 05

Create a new program from the previous lab to print out a description along with the centigrade temperature. For Fahrenheit temperatures in the following ranges, print the corresponding description along with the centigrade conversion:

Temperature:

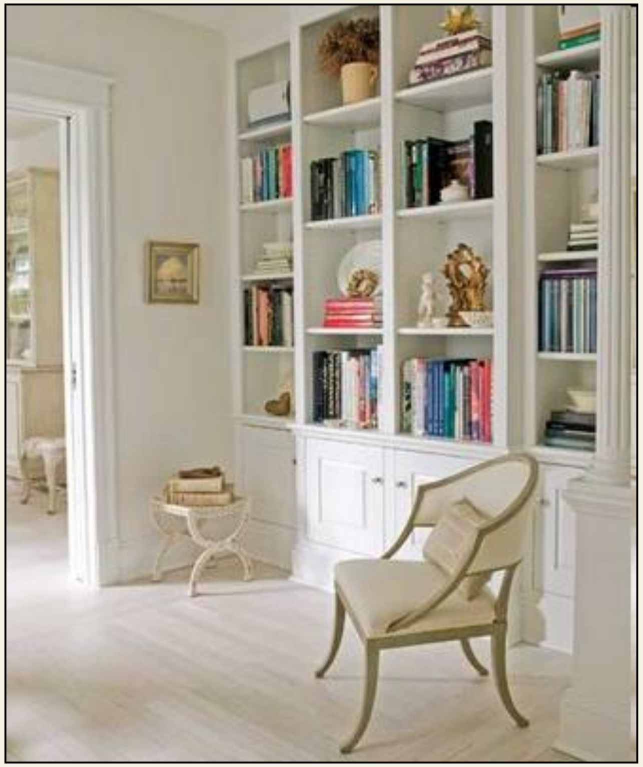
Over 95	It’s very hot!
Over 80 to 95	It’s hot
Over 60 to 80	It’s nice out
Over 40 to 60	It’s chilly
40 or less	It’s cold!

Example: keyboard entry is 62. Program output should be something like:

```
62.0 degrees Fahrenheit is 16.7 degrees Centigrade.
It’s nice out.
```

This exercise requires multiple elif statements following the if statement.

Play it Again Sam (Conditional Iteration)



Reading all books on my bookshelf

```
Repeat while books on shelf I have not read:
   Pick a book that you have not read.
   Read the book.
   Put the book back
 Declare "I have read everything!"
```

Looping ? Times

- Review the concept of iteration.
- One way to loop (or iterate): while statement
- Executes a set of statements in a loop until the while expression turns False, at which time the loop terminates.
- Demo from samples (d1While.jpg)
  - Using a condition or True
  - Use of break

LAB 06

Re-implement the previous lab asking the user for temperatures to convert in a loop.
Stop if raw_input() returns 'q' or 'Q'

Play it Again Sam (Fixed Iteration)

At school as a punishment you might have to write lines: writing the same thing over and over again.

“Write out 100 times, ‘I must not throw chewing gum at the teacher’ ”

“Write out 30 times, ‘I must not break up my desk and pass the bits out of the window behind the teacher’s back’ ”,

Writing Lines

```
1.  Repeat 100 times:
2.      write punishment line
3.      swear silently
```

With fixed iteration we know in advance the number of times an instruction needs to be repeated

Looping N Times

- The for statement – another way to iterate.
  - As with while, there is an else option that is rarely used.
- Executes a set of statements in a loop for the number of times you tell it, executing the same statements with different data.
- Demo using range function.
  - range(first, last, increment)
  - Use of continue
- Sample - d2For.jpg

LAB 07

Re-implement the previous lab using a for statement with a range function to convert to Centigrade the Fahrenheit temperatures from -40 to 110 in increments of ten degrees (i.e., -40, -30, -20, …, 110). Skip the Fahrenheit temperatures zero and 50 when doing the conversions. Use a continue statement to make this happen.

Loops Within Loops

- Loops can contain other loops.
- Example: (Yes, this is nonsense – it's just an example)

```
x = 10
while x > 0:
    for j in range(3):
        print x, j
    x = x - 1
```

- Break and continue statements work only in the loop in which they are executed.
- Break and continue work in both for and while loops.
- Sample - eDual_Loop.jpg

LAB 08

- Create a times table
- It should be a matrix of the results of all possible single-digit multiplications

Ex:

```
1  2  3  4  5  6  7  8  9
2  4  6  8  10 12 14 16 18
```

And so on.

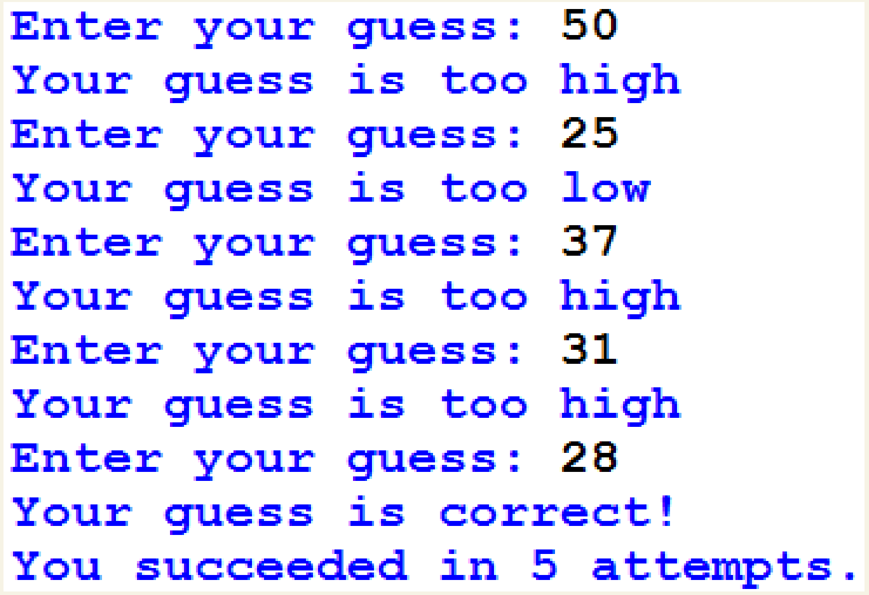
- You should use a for loop inside another for loop.
- Don’t worry about getting numbers aligned unless you finish early.

Importing Modules

- Much of Python’s capability is included in modules.
- You import only what you need.
- Use help to show modules. Interactive shell – help(‘modules’)
- Usually, you will want access to one or more functions within the module.
- Sometimes, you might want access to variables in the module.
- Use pydoc on math, math.sqrt and random.randrange
- Demo with math and random

LAB 09 - Game

Create a game. Have the computer select a random integer between 1 and 100 inclusive. Then, have the operator take successive guesses until they guess the correct number. At each try advise them whether the guess is too high or too low. If the guess is correct, tell them they won and tell them the number of attempts they took. Use the same format as the output sample to the right.



```
Enter your guess: 50
Your guess is too high
Enter your guess: 25
Your guess is too low
Enter your guess: 37
Your guess is too high
Enter your guess: 31
Your guess is too high
Enter your guess: 28
Your guess is correct!
You succeeded in 5 attempts.
```

Functions

- What Python functions have you used already? How did you access them?
- You can build your own functions.
- def/return statements
- Functions are just named groups of statements that return an answer
- Functions are 'called' with arguments
- Functions reduce repetitive code; enable reusability.
- Sample - f1Functions.jpg

LAB 10

LPTHW 19 and 21
http://learnpythonthehardway.org/book/

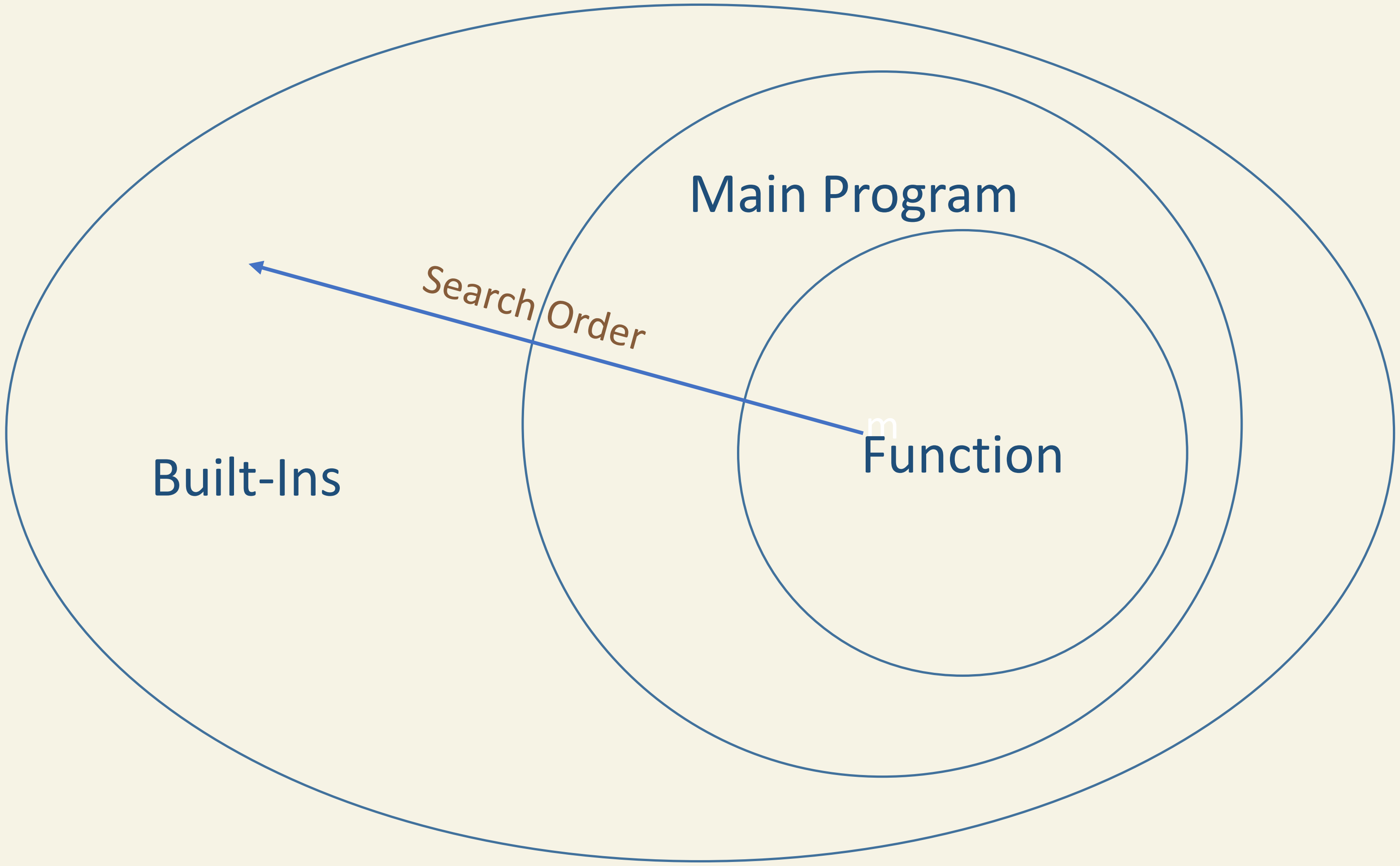
Re-implement the previous lab to use a function to do just the temperature conversion formula.

Function name should be *f_to_c()*

IDENTIFIER SCOPE

- Scope is how widely the identifier can be seen in other parts of the code
- Local scope
- Global scope
- 'Built-in' scope
- Sample - f2Functions global-local.jpg

SCOPE (Simplified)



EXCEPTIONS

- Exceptions are just events: e.g., using an incorrect data type for an operation or using a variable that isn't defined.
- Handling an Exception is optional
- Catching Exceptions requires using a TRY statement
- Some Exceptions: NameError, ValueError, TypeError

CATCHING EXCEPTIONS

- try/except/else/finally
- Puts a protection ring around a block of code
- **except** catches one or more named Exceptions
- Don't use 'bare' **except**
- **else** runs if no Exception occurs
- **finally** always runs no matter what
- Sample - gTry_Except.jpg

LAB 11

Re-implement the previous lab to use a try statement to catch ValueError exceptions. Test by entering a non-numeric temperature.

FILE I/O

The power of open()

File modes

File operations/methods

A file is opened and assigned to a variable.

Every other action is a method acting on the file object.

Demo (open, read, close) with samples iRead_File1.jpg, iRead_File2.jpg and iRead_File3.jpg

LAB 13

Re-implement the previous temperature-conversion lab to replace the data source with a file instead of raw_input(). The file should be the data in the temps.dat file in your data folder. Be sure to account for any bad data that might be contained in this file.

LAB 14 – Read File

Create a plan and write a program to read the file in your data folder labeled trees.dat. This file contains the measurement in even feet of the height of a sampling of California coastal redwood trees. Your job is to read the data and produce a report showing the following information:

The number of trees in the sample,

The average height of all the trees to the nearest tenth of a foot,

The height of the tallest tree,

The height of the shortest tree, and

The number of trees over 300 feet tall

Format the data so the report has a professional look.

LAB 15

The image to the right contains part of LPTHW exercise 16. In place of filename in the open statement, substitute the path/filename on your computer. Then simply enter the code as shown and run it. When finished, open the file you created with a text editor. Eliminate the writing of "\n" from each line of the file and rerun the program. Observe what the output looks like now. What happened to the original data? Why does the new data look this way?

```
print "Opening the file..."
target = open(filename, 'w')
# Where filename contains the full path and name of file

line1 = 'Anydata you want 1'
line2 = 'Anydata you want 2'
line3 = 'Anydata you want 3'

print "I'm going to write these to the file."

target.write(line1)
target.write("\n")
target.write(line2)
target.write("\n")
target.write(line3)
target.write("\n")

print "And finally, we close it."
target.close()
```

LAB 16 – Write File

Change the previous lab on tree heights to include the same report in a file that you can open and read with a text editor. Place this file in the same directory/folder as the input file. Call it whatever you like.

Review

- When writing a program, what is the first thing you do?
- When writing a program, what should you always have?
- What is sequence? What statement is used in selection? What two statements do we use for iteration?
- What is indentation used for in Python? What is the standard indentation?
- What are the three types of errors you can encounter running a program?
- What built-in functions have we used? Imported functions?
- What statement allows us to define our own function? Intercept errors?
- What methods did we use to read a file? Write a file?

LAB 17 - Dice Roll

Use the randrange function in the random module to simulate rolling a pair of dice. Simulate 1000 rolls and calculate the percentage of the rolls that are sevens and the percentage of the rolls that are twos. You can use Python Help or Google to find out more about random and randrange.

If you see the word "self" in the documentation for randrange, completely ignore it for now. Do not even account for it as a parameter when calling the function.

Use Python Help or Pydoc to determine what the difference is between the randrange and randint functions in the Random module.

RESOURCES

- Your humble instructor
- Python for Informatics - book
- learnpythonthehardway.org
- tutorialspoint.com/python
- udacity.com
- Google Code University
- greenteapress.com
- PEP 8 – Python style guide

Python II Overview

- Review including a significant lab.
  - Does not preclude the need for complete understanding of Python I material
- Data structures – purpose, operations and methods
  - Strings
  - Lists, Tuples
  - Dictionaries
  - Sets
- Concept of mutability
- Concept of tables/arrays
- Concept of iterables and iterators

Take-Home Assignment

- Review Python for Informatics (www.pythonlearn.com/book_007.pdf)
  Chapters 1 – 5 and 7 including the exercises – These chapters should be understood completely.
- Preview chapters 6, 8 and 9. Python 2 is very concentrated and intense. You should not take this class until you are at least somewhat familiar with the data structures covered in these three chapters. It is not necessary to master the details or do the exercises. Just be familiar with the subjects prior to class.
- Review and understand all of the completed labs from Python I.

Take-Home LAB

- Create a program that calls a function that simulates the rolling of a pair of dice. Use randrange or randint from the random module to accomplish this. Your main program will deal with the total of the two dice.
- The rules are as follows:
  - On the first roll, a total of 7 or 11 is an automatic win
  - On the first roll, a total of 2, 3 or 12 is an automatic loss
  - Any other number is called the Point. You must roll again. On subsequent rolls the rules are as follows:
    - You roll a 7 which is a loss.
    - You roll the Point number again which is a win.
    - Any other roll has no meaning and another roll is required.
- You start with $100 and bet $10 on each play.
- Print all the rolls and whether you have won or lost on one line.
- Print the funds balance and a request to play again on the next line. A ‘y’ or ‘Y’ means play again. Anything else ends play. A balance of $0 ends play automatically.

Take-Home LAB

The output from your program should look something like the following:

```
Beginning Balance = $100
7  You win!
Balance = $110 – Play again?  y/n:  y
10 4 12 8 7  You lose!
Balance = $100 – Play again?  y/n:  y
3  You lose!
Balance = $90 – Play again?  y/n:  y
8 6 5 9 8  You win!
Balance = $100 – Play again?  y/n:  y
4 12 6 9 7  You lose!
Balance = $90 – Play again?  y/n:  n
Number of plays - 5
Ending Balance = $90
```

THE END