CSC148 Ramp-Up

January 2011
Velian Pandeliev
(based on notes by
Jonathan Taylor & Noah Lockwood)

Overview

In six hours, we'll cover the background required for CSC148

This session is for students with programming experience who haven't necessarily taken the prerequisite, CSC108

Format: 6 modules. First I'll introduce each concept, then you'll do an exercise using the concepts.

Please ask questions!

Python Basics

Comments start with a #:

```
#This is a comment and will not run
```

Dynamically typed

```
x = 3          #CORRECT
int x = 3      #INCORRECT
x = "monkey" #ALSO CORRECT
```

Indentation is significant

```
x = 3
    y = 3 # INCORRECT: no indent needed
```

No extra code needed to start

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

No semicolons at the ends of lines!

Running Python Programs

Python programs are stored in .py files.

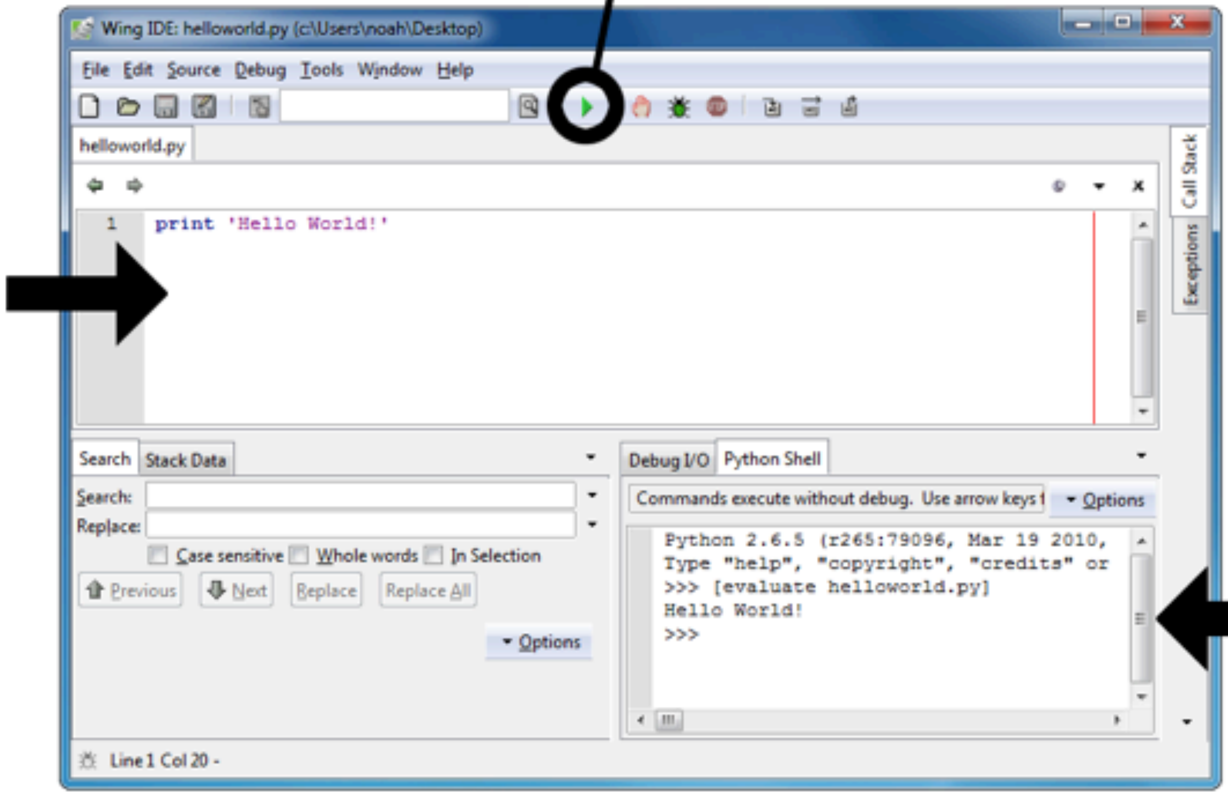
From the command line:

```
#user@redwolf:~$ python helloworld.py
Hello World!
```

Using WING:

Run the current file

Edit python files here



Wing IDE: helloworld.py (c:\Users\noah\Desktop)

```
print 'Hello World!'
```

Commands execute without debug. Use arrow keys f

```
Python 2.6.5 (r265:79096, Mar 19 2010,
Type "help", "copyright", "credits" or
>>> [evaluate helloworld.py]
Hello World!
>>>
```

Line 1 Col 20 -

File output and interactive Python “shell”

Python Reference & Resources

Official Python documentation:

```
http://docs.python.org
```

The `dir` function shows the known methods for a given type:

```
>>> dir(str)
```

The `help` function provides details:

```
>>> help(str)
```

Variables

Integers (`int`):

```
>>> apples = 4
>>> apples / 3
1
```

Floating-point for decimal numbers (`float`):

```
>>> pi = 3.14
>>> radius = 5.0
>>> pi * (radius ** 2)
78.5
```

Boolean (`bool`) for True and False:

```
>>> val = True
>>> not val    # standard Boolean ops: not, and, or
False
>>> 4 > 3      # bool returned by comparison ops
True
```

Strings

```
>>> msg = "Welcome!"
```

Getting a single character in a string:

```
>>> msg[3]  # remember, indices start at 0
'c'
```

Substrings using `[ : ]` ("slicing") notation :

```
>>> msg[4:7]  # characters 4 to 7 - 1
'ome'
>>> msg[3:]  # characters 3 to end of string
'come!'
```

Obtaining the length of the string with built-in `len` function:

```
>>> len(msg)
8
```

Strings can be added together ("concatenated") to form new ones:

```
>>> msg2 = "Come in!"
>>> msg + " " + msg2
"Welcome! Come in!"
```

Strings 2

Operations on string variables using methods:

```
>>> # str.find returns first index of given substring
>>> msg.find('e')
1
>>> msg[1]
'e'
>>> # str.lower returns a lowercase copy of string
>>> msg.lower()
'welcome'
```

Strings are immutable, meaning they can't be changed once created:

```
>>> msg[0] = 'w'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item
assignment
```

Empty strings are OK:

```
>>> msg = ''
```

Input/Output

Reading keyboard input: `raw_input`

```
>>> name = raw_input()
Velian
>>> name
'Velian'
```

Generating Output: `print`

```
>>> print "Hello " + name
Hello Velian

>>> print "Hello %s %s" % ("Chuck", "Norris")
Hello Chuck Norris

>>> print "Name: %s Age: %d Grade %.2f" %
                               ("Bob", 20, 83.33333)
Name: Bob Age: 20 Grade 83.33
```

Conversions

Use the functions `int()`, `float()`, `str()`:

```
>>> float('3.14')
3.14000000001

>>> int('3')
3

>>> float(3)
3.0

>>> str(3.14)
'3.14'
```

But don't try to convert silly things:

```
>>> int("Hello world")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10:
'Hello World!'
```

Exercise 1: Temperature

Write a program that:

- prompts the user for degrees in Fahrenheit

- converts the number into Celsius

- prints out the number in Celsius

The formula for conversion is:

$$C = (F - 32) / 1.8$$

Exercise 1 Solution

```
# Give a prompt
print "Input temperature in fahrenheit:"

# Read in the input.
fahrenheit = raw_input("--> ")

# Convert to floating point.
fahrenheit = float(fahrenheit)

# Calculate the degrees in celsius.
celsius = fahrenheit - 32
celsius = celsius / 1.8

# Display answer.
print "The temperature in %.2f degrees celsius." %
(celsius)
```

List Basics

Lists are a very important data structure in Python

They're created with comma-separated elements in []:

```
>>> colours = ['red', 'green', 'blue']
>>> empty = [] # allowed
>>> mixed = ['red', 3, 5.6]
```

Lists can be indexed:

```
>>> first_colour = colours[0]
>>> first_colour
'red'
>>> colours[0:2]
['red', 'green']
```

Lists can contain anything, even other lists:

```
>>> nested = [1, 2, 3, colours]
>>> nested
[ 1, 2, 3, ['red', 'green', 'blue']]
```

More on Lists

Lists are mutable:

```
>>> colours[2] = 'yellow'
>>> colours
['red', 'green', 'yellow']
>>> colours.append('blue')
['red', 'green', 'yellow', 'blue']
```

```
>>> nums = [4, 2, 1, 3]
>>> nums.sort()
>>> nums
[1, 2, 3, 4]
```

Be careful! Multiple variables may be referring to the same data structure ("aliasing"):

```
>>> orig_list = [0, 1, 2]
>>> copy_list = orig_list
>>> copy_list.append(99)
>>> orig_list
[0, 1, 2, 99]
```

For Loops

Used to repeat something on each element of a list

```
>>> colours = ['red', 'green', 'blue']
>>> >>> for c in colours:
...     print c
...
red
green
blue
```

You can loop through list indices using the `range()` function (`range(x)` returns a list `[0, 1, ..., x-1]`)

```
>>> range(len(colours))
[0, 1, 2]
>>> for i in range(len(colours)):
...     print "Colour %d is: %s" % (i, colours[i])
...
Colour 0 is: red
Colour 1 is: green
Colour 2 is: blue
```

More list processing

Use enumerate() to get the index and element together

```
>>> names = ['Jon Taylor', 'Jill Hearst']
>>> for (n, name) in enumerate(names):
...     print "%d. %s" % (n+1, name)
...
1. Jon Taylor
2. Jill Hearst
```

You can generate a new list with list comprehensions:

```
>>> upper = [name.upper() for name in names]
>>> upper
['JON TAYLOR', 'JILL HEARST']
```

List comprehensions are generally awesome.

Exercise 2: Times Table

Compute a times table for numbers 0-9 as a list of lists.

For example, for numbers 0 to 2, this would be:

```
[[0, 0, 0], [0, 1, 2], [0, 2, 4]]
```

Exercise 2 Solution

```
times_table = []
n = 10

for i in range(n):
    # Compute a row
    row = []
    for j in range(n):
        row.append(i * j)
    # add row to timestable
    times_table.append(row)
```

And here's a solution with list comprehensions:

```
[[i*j for i in range(n)] for j in range(n)]
```

But resist the temptation to overuse them.

If Statements

If statements allow you to make decisions based on whether a certain condition is True or False:

```
if amount > balance:
    print "Not enough money!"

elif amount == balance:
    print "You must keep a positive balance!"

else:
    amount = amount - balance
    print "Transaction OK"
```

elif stands for 'else if', elif and else are both optional:

```
if amount >= balance:
    print "Transaction NOT OK"

else:
    amount = amount - balance
    print "Transaction OK"
```

Functions

Remove duplication of code and to encapsulate commonly used sequences of commands:

```
def celsius_to_fahrenheit(degrees):
    celsius = float(degrees)
    fahrenheit = (1.8 * celsius) + 32
    return fahrenheit

f = celsius_to_fahrenheit(0)

def print_list(list):
    if len(list) == 0:
        print "List is empty!"
    else:
        print "List contents:"
        for x in list:
            print x
```

Exercise 3: Functions

Two words are a reverse pair if each word is the reverse of the other.

- Write a function `is_reverse_pair(s1, s2)` that returns `True` if and only if `s1` and `s2` are a reverse pair.

- Then, write a function `print_reverse_pairs (wordlist)` that accepts a list of strings and prints out all of the reverse pairs in the list.

Exercise 3 Solution

```
  def is_reverse_pair(s1, s2):
    if len(s1) != len(s2):
        return False
    for i in range(len(s1)):
        if s1[i] != s2[len(s2) - 1 - i]:
            return False
    return True
```

Or, using slicing notation:

```
def is_reverse_pair(s1, s2):
    return s1[::-1] == s2

def print_reverse_pairs(wordlist):
    for s1 in wordlist:
        for s2 in wordlist:
            if is_reverse_pair(s1, s2):
                print ‘%s, %s’ % (s1, s2)
```

Tuples

A faster, simpler way to represent a collection of objects. Like lists, but immutable (meaning what?)

```
>>> seq = (4, 'f', 'foo', 2)
>>> seq
(4, 'f', 'foo', 2)
```

Tuples can be converted to lists:

```
>>> l = list(seq)
```

Caveat: Tricky to define a one element sequence:

```
>>> seq = (1)
>>> seq
1
>>> seq = (1,)
>>> seq
(1,)
```

Dictionaries

Dictionaries associate elements with keys rather than indices. They contain key-value pairs, defined as follows:

```
>>> scores = {'Alice': 80, 'Bob': 70, 'Eve' : 80 }
>>> scores['Bob']
70
>>> scores['Dave'] = 90 # adds pair to dictionary
>>> scores
{‘Dave’: 90, ‘Bob’: 70, ‘Alice’: 80, ‘Eve’: 80}
```

```
>>> scores.keys() # list of keys
['Dave, 'Bob', 'Alice', 'Eve']
```

```
>>> scores.items() # list of pairs
[('Dave', 90), ('Bob', 70), ('Alice', 80), ('Eve', 
80)]
```

Note: Keys in dictionaries have to be unique, and must be immutable.

Note 2: Dictionaries do NOT maintain order of elements.

Exercise 4: Dictionaries

- Write a function `print_record` that takes a dictionary as input. Keys are student numbers (`int`), values are names (`str`). The function should print out all records, nicely formatted.

```
>>> record = {1234 : ‘Tony Stark’, 1138 : ‘Steve
Rogers’}
>>> print_record(record)
Name: Tony Stark
Student #: 1234

Name: Steve Rogers
Student #: 1138
```

- Write a function count_occurrences that takes a list of strings as input, and returns a dictionary with key/value pairs of each word and the number of occurrences of that word.

```
>>> count_occurences([‘a’, ‘b’, ‘a’, ‘a’, ‘c’, ‘c’])

{‘a’ : 3, ‘b’: 1, ‘c’: 2}
```

Exercise 4 Solution

```
  def print_record(rec):
    for (num, name) in rec.items():
        print ‘Name: ’ + name
        print ‘Student #: ’ + num
        print ‘’

def count_occurrences(words):
    result = {}
    for word in words:
        if word in result.keys():
            result[word] = result[word] + 1
        else:
            result[word] = 1
    return result
```

While Loops

A while loop is used to repeat a sequence of statements as many times as is necessary as long as a particular condition is True:

```
count = 1
while count <= 10:
    print count
    count += 1
```

You can break out of a while loop using the `break` statement.

```
count = 1
while True: # this is an infinite loop
    print count
    count += 1
    if count == 10:
        break # and this is how to get out of it
```

Modules

Python comes with many modules that provide useful functionality:

```
>>> import random
>>> random.randint(1,6) # roll a die
5

>>> import math
>>> math.sqrt(8)
2.8284271247461903
>>> math.cos(1)
0.5403023058681397
>>> math.cos(0)
1.0

>>> from datetime import date
>>> date.today()
datetime.date(2012, 1, 7)
```

Exercise 5: Guessing Game

Implement a number guessing game:

```
Guess a number between 0 and 100:
--> 50
Too High.
Guess a number between 0 and 100:
--> 25
Too High.
Guess a number between 0 and 100:
--> 13
Too High.
Guess a number between 0 and 100:
--> 8
Correct.
```

Optional: set a 5-guess limit.

Exercise 5 Solution

```
import random
# Choose a random number.
num = random.randint(0,100)

found = False
while not found:
    print "Guess a number:"
    guess = int(raw_input())
    if guess > num:
        print "Too High."
    elif guess < num:
        print "Too Low."
    else:
        print "Correct."
        found = True
        # or you could try
        # break
```

Classes and Objects

Classes are used to organize data and provide special operations. We will talk more about these in CSC148, this is just a primer.

```
class Person(object):
    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name
    def __str__(self):
        return self.first_name + " " + self.last_name
>>> p = Person('Jon', 'Taylor')
>>> p
<person.Person object at 0xb7cf720c>
>>> print p
Jon Taylor
```

Exercise 6: NumberList

Write a class that stores a list of integers/floats and provides the following methods:

`sum()` - returns the sum of the list

`mean()` - returns the average of the list as a float

`min()/max()` - returns the maximum/minimum element

`num_uniques()` - returns the number of unique elements in the list

Hint: Use the `in` keyword:

```
>>> nums = [1, 3, 9, 16]
>>> 3 in nums
True
>>> 7 in nums
False
```

Exercise 6 Solution

```
class NumberList(object):

    def __init__(self, l):
        self.l = l

    def sum(self):
        cur = 0.0
        for x in self.l:
            cur = cur + x
        return cur

    def mean(self):
        n = len(self.l)
        sum = self.sum()
        return float(sum)/n
```

Exercise 6 Solution

```
...

    def max(self):
        cur = self.l[0]
        for x in self.l:
            if x > cur:
                cur = x
        return cur

    def num_uniques(self):
       count = 0
       for i in range(len(self.l)):
           if not self.l[i] in \
                       (self.l[:i] + self.l[i+1:]):
               count = count + 1
       return count
```