AMACSS PRESENTS…

CSCA08 FINAL REVIEW SEMINAR

BOOLEAN EXPRESSIONS

- True, False
- Evaluation of Boolean Expressions
  e.g. x = (3>5)
- Operators
  - AND (`True and False`)
  - OR (`True or False`)
  - NOT (`not True`)

CONDITIONAL STATEMENTS

- Check if some condition(s) have been satisfied then run the code below it
- if(some_condition_to_be_checked)… then… else…

```
if ( x > y ):
 print(x)
elif ( x == y ):
 print(“same”)
else:
 print(y)
```

COMMON PYTHON DATA TYPES

- *String* s = “Hello and Goodbye”
- *Char* c = ‘x’
- *Int* i = 5
- *Float* f = 2.0
- *Double* d = 2.49
- *Boolean* b = False

PYTHON DATA STRUCTURES — LISTS

```
l = []

l.append(item)

l.pop(item)

print(l)

l[index]

l[start:end]
```

- Lists can be of any type – integer, string, object, etc. - can be mixed!
- We can represent Strings as Lists – use common list operations to work with them

LIST REPRESENTATION OF STRINGS

- Think about a string as a list of characters

```
string = “ZA” = [‘Z’, ‘A’]
```

We can perform normal list operations for the most part!

```
print(string[0])

print(string + “ WARUDO”)
```

LIST REPRESENTATION OF STRINGS

- Think about a string as a list of characters

```
string = “ZA” = [‘Z’, ‘A’]
```

We can perform normal list operations for the most part…

```
c = ’daze’

c[0] = ’y’

print(c[0])
```

LIST REPRESENTATION OF STRINGS

- Think about a string as a list of characters

```
string = “ZA” = [‘Z’, ‘A’]
```

We can perform normal list operations for the most part…

```
e = [’A’, ’B’, ’C’]

e[0] = 8663

print(e)
```

LOOPS

- While loops - continues on while the condition is true
  - Good for when we dont know how many operations we want to perform

```
while(True):

while(condition):
```

- For loops - continues on for some x amount of times
  - Good for when we know exactly how many operations we want to perform

```
for i in l:

for i in range(0, x):
```

FUNCTIONS

- Functions are like math functions, take some input, does some stuff, returns some output

```
def add_one(x):
  return x+1

def stutter(x):
  stuttered_word = []
  for i in x:
    stuttered_word.append(i)
    stuttered_word.append(i)
  return stuttered_word
```

USING FUNCTIONS, LOOPS, AND STRINGS

```
def differ_by_one(word1, word2):
    ’’’ (str, str) -> bool
    Return True iff word2 can be formed from word1 by changing exactly one letter.
    >>> differ_by_one(’cat’, ’cot’)
    True
    >>> differ_by_one(’abc’, ’aBc’)
    True
    >>> differ_by_one(’abc’, ’abc’)
    False
    >>> differ_by_one(’abc’, ’abcd’)
    False
’’’
```

QUESTION

APPLYING LOOPS AND STRING MANIPULATION

```
def differ_by_one(word1, word2):
    result = True
    dif = 0
    if(len(word1) != len(word2)):
        result = False
    else
      for i in range(0, len(word1)):
         if(word1[i] != word2[i]):
             dif += 1
      if(dif != 1):
         result = False
    return result
```

PYTHON DATA STRUCTURES — TUPLES

```
l = ()

l[index]

l[start:end]

e.g. l = (1, 2, “Three”, (4 == “Four”))
```

- Tuples can be of any type – integer, string, object, etc. - can be mixed!
- Why use Tuples? **Immutability…**

PYTHON DATA STRUCTURES — SETS

```
S = set()

x in S

s1.union(s2)

e.g. S = set(3, 4, 5)
```

- Sets can be of any type – integer, string, object, etc. - can be mixed!
- Interesting to note: **Sets have no ordering.**
- Why use Sets? **Existence Checks. We also cannot insert more than one copy of any element into one set.**

PYTHON DATA STRUCTURES — DICTIONARY

```
D = {}

d[key] = value
```

- Dictionaries associate values to different keys
- We find this useful if we want to store values that are mapped to keys. However, dictionaries have **no ordering.**
- Why use Dictionaries? **Create mappings.**

PYTHON - CLASSES

- A class in Python defines a specific object or a set of methods that have relations to each other

```
class Triangle():
    def __init__(self, h=None, b=None):
        self.height = h
        self.base = b

    def get_area(self):
        return (self.height * self.base)/2

    def __str__(self):
        return "Triangle with base: " + str(self.base) + " and height: " +
str(self.height);
```

```
>>> m = Triangle(4, 5)
>>> m.get_area()
10.0
>>> print(m)
Triangle with base: 5 and height: 4
```

PYTHON - CLASS INHERITANCE

```
class GrandParent():
     def __init__(self, a, b):
         self._a = a
         self._b = b
     def blah(self):
         return "GP:" + self._a + self._b
 class Parent1(GrandParent):
     def __init__(self, a, b, c):
         GrandParent.__init__(self, a, b)
         self._c = c
     def blah(self):
         return ("P1:" + self._a +
                 self._b + self._c)
```

Initial - Parent Class

Super Constructor

Method Overriding

PYTHON - CLASS INHERITANCE

```
class GrandParent():
     def __init__(self, a, b):
         self._a = a
         self._b = b
     def blah(self):
         return "GP:" + self._a + self._b
 class Parent1(GrandParent):
     def __init__(self, a, b, c):
         GrandParent.__init__(self, a, b)
         self._c = c
     def blah(self):
         return ("P1:" + self._a +
                 self._b + self._c)
```

```
>>> gp = GrandParent("A", "B")
>>> print(gp.blah())
GP:AB
>>> p1 = Parent1("A", "B", "C")
>>> print(p1.blah())
P1:ABC
```

PYTHON - CLASSES AND SCOPE

- Variables have a scope – where we can see, use, and reference them. We usually choose the variable with the most local scope to use.

```
def scopeTest(x):
    x = 5
    y = 100
    print(x)

x = 9
print(x)
print(scopeTest(x))
print(y)
```

```
9
5
None
Traceback (most recent call last):
  File “filepath…/”, line 61, in <module>
    print(y)
builtins.NameError: name 'y' is not defined
```

COMPLEXITY AND ANALYSIS

- Complexity analysis the running time of a program with relation to its input, typically called n.
- We have different runtime categorizations, such as
  - `O(1)`
  - `O(n)`
  - `O(n^2)`
  - `O(log n)`
  - `and more… (e.g. O(n^2), O(n!), O(2^n)…)`
- We call these the worst-case complexities – the worst case scenarios of our program(s) when we get input of size n

BREAK AND QUESTIONS

- After the break we’ll move towards solving some questions on the past final(s)
- Please tell us if you have any question about course material, past final challenging questions etc.
- Leave us a review on http://amacss.org

CODE SHOWN IN REVIEW SEMINAR

```
def differ_by_one(x, y):
    result = True
    dif = 0
    if(len(x) != len(y)):
        result = False
    for i in range(0, len(x)):
        if(x[i] != y[i]):
            dif += 1
    if(dif != 1):
        result = False
    return result

def stutter(x):
    s = []
    for i in x:
        s.append(i)
        s.append(i)
    return s

class Triangle():
    def __init__(self, h=None, b=None):
        self.height = h
        self.base = b

    def get_area(self):
        return (self.height * self.base)/2

    def __str__(self):
        return "Triangle with base: " + str(self.base) + " and height: " + str(self.height);

class GrandParent():
    def __init__(self, a, b):
        self._a = a
        self._b = b
    def blah(self):
        return "GP:" + self._a + self._b

class Parent1(GrandParent):
    def __init__(self, a, b, c):
        GrandParent.__init__(self, a, b)
        self._c = c

    def blah(self):
        return ("P1:" + self._a + self._b + self._c)

class Child(Parent1, GrandParent):
    def __init__(self):
        pass

class Parent2(GrandParent):
    def __init__(self, a, b, c):
        self._a = b
        self._b = a
        self._c = self.blah()

class Child1(Parent2):
     def __init__(self, a, b, c, d):
         Parent1.__init__(self, a, b, c)
         self._d = d

def scopeTest(x):
    x = 5
    y = 100
    print(x)
```