LECTURE 3: MORE PYTHON

CONDITIONALS

- A python program:

```
print('The exam had 40 points’)
score = int(input("What was your score?"))
percent = 100 * score / 40.0
print('Your score in percentage was', percent)
if percent>= 90.0:# Note the syntax here, end with :
     print("Congrats, you got an A”) #indentation is
significant
```

- If value is larger than or equal to 90, print a congrats message, otherwise, no statement is printed

INDENTATION

- So far all statements have started in the first column
- In this program, the statement following the if statement is indented
- Indentation is the leading whitespace(space sand tabs) at the beginning of a line
- Multiple statements of the same indentation belong to the same group
- There can be multiple levels of indentation
  A tab is treated as 8 spaces in Python2, but not Python3

ELSE STATEMENT

-
  ```
  if percent>= 90.0:
      print("Congrats, you got an A”)
      print("You are doing well in this class”)
  print("see you in class next week”)
  ```

- If value is larger than 90, print two statements. In either case, print a see you in class statement.

-
  ```
  if percent>= 90.0:
      print("Congrats, you got an A”)
      print("You are doing well in this class”)
  else:
      print("you did not get an A”)
  print("see you in class next week”)
  ```

  If value is higher than 89, print two statements. Otherwise, print one statement. Afterwards, in both cases, print a see you in class statement

NESTED CONDITIONALS

- If statements can be nested inside each other. We must pay special attention to the indentation. e.g.,

```
if percent>= 90.0:
    if percent>= 95.0:
        print('you get an A+!’)
    else:
        print('you get an A’)
```

```
if percent>= 90.0:
    if percent>= 95.0:

        print('you get an A+!’)
else:

    print('you get an A’)
```

- Any problem here?

THE ELIF STATEMENT

```
print('The exam had 40 points’)
score = int(input("What was your score?"))
percent = 100 * score / 40.0
print('Your score in percentage was', percent)

if percent>= 90.0:

    print('Congrats, you got an A’)

elif percent>= 80.0:

    print('you got a B’)

elif percent>=70.0:
    print('you got a C’)

else:

    print('your grade is lower than C’)
```

WHILE LOOPS

```
sum = 0.0
count = 0
num = int(input("Enter your number: "))
while num != -1:
    sum = sum + num
    count = count + 1
    num = int(input ("Enter your number: "))
print("average is", sum / count)
```

- What if the user enters -1 the very first time?

THE BREAK STATEMENT

break: terminates the current loop in the middle of its execution; breaks out of the current loop

```
sum = 0.0
count = 0
while True:
    num = int(input("Enter your number: "))
    if num == -1:
        break
    sum = sum + num
    count = count + 1
print("average is", sum / count)
```

FOR LOOPS

```
for num in range(1,8):
    rslt = num * 10
    print('number is', num, 'result is', rslt)
```

- range(1,8) denotes the range [1,8), i.e., 1 to 7
- range(10) denotes the range [0,9], i.e., 0 to 9
- range(0,10,2) iterates with a stride of 2

```
for ch in 'abacadabra’:
    if ch in'aeiou’:
        print('letter', ch, 'is a vowel’)
    else:
        print('letter', ch, 'is not a vowel’)
```

THE CONTINUE STATEMENT

- continue: ends the current loop in the middle of its execution, immediately returns to the condition test of the loop

```
for num in range(2, 6):
    if num%2==0:
        print("Found an even number", num)
        continue
    print("Found a number", num)
```

FUNCTION

- Function blocks begin with the def keyword, followed by the function name and parentheses
- Code within a function starts with ":" and is indented
- The optional "return [expression]" exits a function and returns a value
- Fruitless vs fruitful functions

```
def area_of_rect(w,h):
    return w * h
print('the area of the rectangle 2 by 3 is ', area_of_rect(2,3))
```

VOLUME OF A SPHERE

```
import math
def vol_of_radius(r):
    ’’’Computes the volume of a sphere’’’
    return 4 * math.pi * r * r * r / 3.0
```

- Try with different values
- Try typing ‘help(vol_of_radius)’

FRUITLESS FUNCTIONS

```
def main():
    # Celsius to Fahrenheit conversion
    # written by Robin Smith, July 2007
    print("We convert a temperature”)
    print("in C to the equivalent in F”)
    c = int(input("Your temperature in C:"))
    f = c * 9.0 / 5.0 + 32
    print("The temperature in F is", f)
```

- Notice the lack of return?

NAME SCOPES

```
>>> x = 4
>>> def scope_test(a):
        return x+a
>>> scope_test(2)
>>> print('x is', x)
x is 4
```

The function creates a *local scope*, outside is a *global scope*. When searching for meaning of a name, python starts from the innermost box where the name appears and looks outward until it finds a matching name.

RECURSIVE FUNCTIONS

- A function that calls itself is called a recursive function
- Any recursive function can be divided into two parts:
  - Base case(s): Where we handle the most basic case
  - Recursive case(s): Where we reduce the problem to a simpler problem of the same form
- Advantages?
  - Intuitive, elegant, appears in job interviews
- Disadvantages?
  - Inefficient as the problem size grows

RECURSIVE EXAMPLE: FACTORIAL

- Write the recursive function factorial in Python
- Base case: 0! and 1!
- Recursive case: n * (n-1)!

DATA STRUCTURES

- Lists
- Tuples
- Dictionaries
- Sets

LISTS

- A list is created inside square brackets
- Not necessarily of the same type

```
lst = [4,2,1,6,19]
lsttwo = [5,’hello’, lst]
```

BASIC OPERATIONS ON LISTS

```
>>> lst
>>> lst[2]
>>> lst[1:3]
>>> lst[2:]
>>> lst[:2]
>>> 4 in lst
>>> 5 in lst
>>> lst + [3,4]
>>> len(lst)
>>> [2,3] * 4
>>> max(lst)
>>> min(lst)
>>> list('123')
```

LISTS ARE MUTABLE

- A list is mutable, elements or the slices can be changed
- del statement can be used to remove elements from a list
- The substitute can be of different size

```
>>> lst = [1, 4, 7, 9, 12]
>>> lst[1] = 5
>>> lst
[1, 5, 7, 9, 12]
>>> lst[1:3] = [2,3]
[1, 2, 3, 9, 12]
>>> del lst[2]
[1, 2, 9, 12]
```

```
>>> lst[1:3] = [9, 8, 7, 6]
>>> lst
[1, 9, 8, 7, 6, 9, 12]
```

MORE BUILT-IN FUNCTIONS

Operation	**Description # lst = [1, 7, 3, 9, 2]**
s.append(x)	lst.append(4) # [1, 7, 3, 9, 2, 4] – growing the list
s.extend(ls)	lst.extend([8,5]) # [1, 7, 3, 9, 2, 4, 8, 5]
s.count(x)	lst.count(5) # 1 - number of occurrences of x
s.index(x)	lst.index(5) # 7 – the index of the first occurrence of x
s.pop()	lst.pop() # 5 – return and remove the last element
s.pop(i)	lst.pop(2) # 3 – return and remove the ith element
s.remove(x)	lst.remove(4) # search for x and remove it (1st one only)
s.reverse()	lst.reverse() # [8, 2, 9, 7, 1] - reverse elements in place
s.sort()	lst.sort() # [1, 2, 7, 8, 9] – sort elements in ascending
s.insert(i,x)	lst.insert(3,2) # [1, 2, 7, 2, 8, 9] – insert x at location i

EXAMPLE: FINDING AVERAGE AND STANDARD DEVIATION

- Three ways of calculating average (Among many):
  - Use a loop
    - Add then divide by the length
    - Divide by the length, then add
  - Use the built-in sum function
- Which one is the best?

$$AM = \frac{1}{n}\sum_{i=1}^{n} a_i = \frac{1}{n}\left(a_1 + a_2 + \cdots + a_n\right)$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \text{ where } \mu = \frac{1}{N}\sum_{i=1}^{N} x_i.$$

LISTS VS STRINGS

- Strings are immutable, lists are mutable
- Hence, to modify a string, convert it to a list, modify it, and convert it back

```
>>> list('abc')
['a', 'b', 'c']
>>> "".join(['a', 'b', 'c'])
'abc'
```

FIXED SIZE AND NESTED ARRAYS

```
>>> data = [0]*5

>>> data

[0, 0, 0, 0, 0]

>>> for i in range(5): data[i] = [0]*5

>>> data

>>> [[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0,
0, 0], [0, 0, 0, 0, 0]]
```

LIST ASSIGNMENTS AND REFERENCING

```
>>> a = [1,2,3]
>>> b = a
>>> b[1] = 7
>>> a
[1, 7, 3]
```

- Use true copy!

```
>>> a = [1,2,3]
>>> b = a[:]
```

FIN!