**PYTHON SUMMARY** (as of 4-21-2011)

## Comment Statements

- begins with the # symbol
- # can be at the start of a line (and thus the whole line a comment line)
- or # can be within a line (and thus the comment is from # to the end of the line)
- comments used at the start of a program to describe it to the reader of the program
- also used within the program to make the major sections
- can be used within a function to describe what it does and how it should be called
- EXAMPLES:

```
# this is a complete comment line
balance = 1000   # init account balance to $1,000
```

## Literal Values

### Numeric Literals

Integers

- no commas allowed
- must not contain a decimal point
- no limit in the size of integers
- EXAMPLE: 1024

Floating Point Values

- no commas allowed
- must contain a decimal point
- limit to the range and precision
- EXAMPLES: 1204.0 1204.42

### String Literals

- can use single or double quotes
- can contain any characters (digits, letters, special symbols)
- can contain single quotes if surrounded by double quotes (and vice versa)
- string operators: + (to concatenate two strings together), len(str) to get length of string
- accessing individual characters in a string using the same indexing as for lists
- EXAMPLES: 'Hello' "Hello" "Let's Begin" "4th quarter profits" "** Congratulations **"

```
'Hello' + name   (concatenates the two, where name contains a string)
first_initial = first_name[0]   # gets the first letter of the name in first_name
```

### Boolean (Logical) Literal Values

- only two possible values: True False
- look like variables names, but are literal values

## Identifiers

- names that the programmer used for variable and function names
- must not begin with a digit, but can contain digits
- can also contain the special underscore (_) character
- can be essentially any length
- EXAMPLES: n n1 num1 yearly_sales total_sales_2011

## Variable Assignment

- use the = symbol for assignment (== is for comparison, not assignment)
- can assign a variable to a single value, or to an expression (that evaluates to a single value)
- the first time that a variables is assigned a value it is defined (created)
- EXAMPLES: n = 10 n = k * 12 n = input('Enter your age:')

## Operators

### Arithmetic Operators

+ (addition), - (negation, subtraction), * (multiplication), / (division), % (modulus)

- the / operator performs integer division is both operands are integers
- if at least one of the operators is a float, then the / operator performs real division

- the modulus (%) operator:

  0 % 10 → 0 1 % 10 → 1, …. , 9 % 10 → 9,
  10 % 10 → 0, 11 % 10 → 1, …., 19 % 10 → 9, …

- EXAMPLE: 2024 / 100 → 2024 % 100 → 24 (a way to split a number into two parts)

### Relational Operators

- < (less than), > (greater than), <= (less than or equal to), >= (greater than or equal to), != (not equal to), == (equal to)

- EXAMPLES: 5 < 10 → True 10 <= 5 → False 10 != 5 → True 10 == 10 → True

### Boolean Operators

- x and y (both x and y must be true for this expression to be True)
- x or y (at least one of x and y must be true for this expression to be True)
- not x (this is True if x is False)
- EXAMPLES: 5 < 10 and 6 < 12 → True 5 < 10 or 8 < 4 → True not 5 > 10 → True

## Expressions

- all expressions evaluate to a value
- can use parentheses to indicate how an expression is to be evaluated,
  otherwise the rules of operator precedence apply (* and / before + and -)
- thus, they can be used wherever the type of value that they evaluate to can be used

### Arithmetic Expressions

- any combination of numeric values and arithmetic operators
- EXAMPLES: num * 20 num1 * (20 / num2) – num3

### Relational Expressions

- any combination of ordered values and relational operators
- EXAMPLES: 10 < 20 'a' < 'z' 10 == 20

### Boolean Expressions

- any combination of Boolean values/Boolean and relational operators
- EXAMPLES: 1 < 10 and 4 > 2
  married and has_child (where variables married an has_child are Boolean variables)

## Input/Output

### Input

- use input() for reading numeric values from user
- use raw_input() for reading string values from user (such as a name)
- EXAMPLES:

```
age = input('Enter your age:')
name = input('Enter your name:')
```

### Output

print 'Hello'
- prints 'Hello' and moves screen cursor to next line

print 'Hello',
- prints 'Hello' and leaves cursor on the same screen line
- thus, next print output will begin where previous one left off on screen

print 'The result is', result
- can print a combination of literal values and variables

## Control

### Selection

**if statement without else**

```
if age < 14:
      print   "YOU'RE JUST A KID"
```

**if statement with else**

```
if age < 14:
    print   "YOU'RE JUST A KID"
else:
    if age < 22:
        print "We're about the same age"
    else:
        print "You're older than me"
```

"Catch all" (optional)

**if statement with chained elif**

```
if numCredits >= 90:
   print 'Senior'
elif numCredits >= 60:
   print 'Junior'
elif numCredits >= 30:
   print 'Sophomore'
else
   print 'Freshman'
```

"Catch all" (optional)

### Repetition (Iteration – while loops and for loops)

```
num = 1
sum = 0
while num <= 100:
    sum = sum + num
    num = num + 1
```

**Definite loop**
Adds up first 100 integers
Same as repeat(100) in Scratch

```
sum  = 0
for num in range(1,101):
    sum = sum + num
```

**Definite loop**
Logically equivalent to above using a for loop instead of a while loop

```
num = input('Enter a number between 1-10, inclusive')
while (num < 1)   or   (num > 10):
    num = input('Enter a number between 1-10, inclusive')
```

```
num = input('Enter a number between 1-10, inclusive')
while not ((1 <= num) and (num <= 10)):
    num = input('Enter a number between 1-10, inclusive')
```

**Indefinite loop**
Logically equivalent using different Boolean expressions

## Lists

### Simple Lists (list of literal values)

```
[]                - empty list

[1,2,3]           - list of length three

nums = [1,2,3]    - assigning a list to a variable

nums[0]           - accesses the first item of list nums (1)
nums[1]           - accesses the second item of list nums (2)

len(nums)         - gives the length of list nums (3)
```

### Natural Use of for Loop with Lists

```
sum = 0
numItems = len(nums)

for k in range(0, numItems):
    sum = sum + nums[k]
```

Adds up all the numbers in list nums, where nums can be a list of any length

### Nested Lists (list of lists)

```
lst = [ [1,2,3], [4,5,6], [7,8,9] ]
```

lst[0] → [1,2,3]  lst[1] → [4,5,6]  lst[2] → [7,8,9]

lst[0][0] → 1  lst[0][1] → 2  lst[0][2] → 3
lst[1][0] → 4  lst[1][1] → 5  etc.

```
sum = 0
for k in range (0, len(lst)):
    for j in range(0,3):
        sum = sum + lst[k][j]
```

Adds up all the numbers in lst

```
for k in range(0, len(lst)):
    for j in range(0,3):
        print lst[k][j],     # comma used to keep cursor on same line
    print                    # "empty print" to move cursor to next line
```

Prints items in three rows, three numbers per row

## Functions

- variables assigned within a function are called “local variables”
- local variables only exists for the function that they are part of
- functions cannot access the local variables of other functions

**Value-Returning Functions**

- can be given 0 or more parameters
- must contain a return statement
- can be called from whereever the return value can be appropriately used
- called as part of an expression, an assignment statement, a print statement, etc.

**# example of a value-returning function with parameters**

```
def avg(n1, n2, n3):
    result = (n1 + n2 + n3) / 3.0
    return result
```

```
def avg(n1, n2, n3):
    return (n1 + n2 + n3) / 3.0
```

**# example of a value-returning function with no parameters**

```
def getInput():
    selection = input('Enter B, D, W, or Q to quit: ')
    while selection != 'B' and selection != 'D' and selection != 'W':
        print '* Invalid Response – Please Reenter *'

    return selection
```

**Non Value-Returning Functions**

- can be given 0 or more parameters
- do NOT contain a return statement
- are NOT called as part of an expression, return statement or print statement since they do not return a value
- cause some other “side effect” such as printing to the screen

**# example of a non-value returning function with no parameters**

```
def welcomeScreen()
    print 'Welcome to the ATM simulation program'
    print '----------------------------------------------------'
    print ' This program has the following options:'
    print  ' B - to check account balance'
    print  ' D - to make a deposit'
    print  'W- to make a withdrawal'
```

```
# example of a non-value returning function with parameters
    def welcomeScreen(name)
        print 'Welcome', name, ' to the ATM simulation program'
        print '------------------------------------------------------'
        print ' This program has the following options:'
        print  ' B - to check account balance'
        print  ' D - to make a deposit'
        print  'W- to make a withdrawal'


    # example of a value returning function passed a list as a parameter
    def total(lst)
        sum = 0   # local variable
        for k in range(0, len(lst)):
            sum = sum + lst[k]

        return sum


    # example main program using above functions

        list1 = [10,45,30,67,52,30,19]

        sum_list1 = total(list1)
        print 'The total of all items in list1 is:', sum_list1
```

OR

```
        print 'The total of all items in list1 is:', total(list1)


        list2 = [ [20,42,53,76,32,42,19], [23,53,48,56,34,32], [23,4,43] ]

        sum_list2 = total(list2[0]) + total(list2[1]) + total(list2[2])
        print 'The total of all items in list2 is:', sum_list2
```

OR

```
        print 'The total of all items in list2 is:', total(list2[0]) + total(list2[1]) + total(list2[2])
```