Chapter 1: Scalar Variables and Data Types

1. Python as a calculator

The Python interpreter acts as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators +, -, * and / work just like on your regular calculator; parentheses can be used for grouping. For example:

```
>>> 1+3
4
>>> # This is a comment
>>> 2+2  # and a comment on the same line as code
4
>>> (60-5*6)/3
10
>>> 7/3 # Integer division returns the floor:
2
>>> 7/-3
-3
```

Remember that, by default, Python only has a limited set of keywords. For example, it only knows how to do the basic mathematical operations (+,-,/,x). If you want a more scientific calculator, you need to first import the math functions included in the module "math":

```
From math import *
```

2. Python Variables

A variable is a name reference to a memory location. Variables provide an easy handle to keep track of data stored in memory. Most often, we do not know the exact value of what is in a particular memory location; rather we know the type of data that is stored there.

Python has three main types of variables:

- Scalar variables hold the basic building blocks of data: numbers, and characters.
- Array variables hold lists referenced by numbers (indices)
- Dictionary variables hold lists references by labels.

The name of a variable can be practically any combination of characters and of arbitrary length. Note that the type of a variable cannot usually not be guessed from its name: I strongly advise

you to choose a name for a variable that makes this type explicit. For example you can use names like X, X_list, X_dic to define a scalar, a list, and a dictionary, respectively.

There are a few rules regarding variable names that you need to be aware of:

- The first character of the name of a variable cannot be a digit
- Spaces are one type of characters that are not allowed: use underscore instead.
- Variables are case sensitive: this means that abc refers to a different location in memory than ABC.

Creating a variable is as simple as making up a variable name and assigning a value to it.

Assigning a value to a variable is easy: all you have to do is write an equation, with the variable name on the left, an = sign, and the value on the left. The = sign is called the assignment operator:

```
>>>Width=4                          # Note that the value of an assignment is not written
>>>Height=3*12
>>>Area=Width*Height
>>>print Area
144
>>>x=y=z=0                          # Python allows multiple assignments: x, y and z are
                                       set to 0
>>>DNA='aattgcg'                    # assign a string variable
>>>Name_list=['John','David']       # set up a list of names
```

3. Special variable

Python has one special variable, _, that points to the place in memory that stores the more recent result:

```
>>> 4+5
9
>>>print _
9
```

This special variable "_" should be considered as "read-only", i.e. I strongly advise against assigning a value to it!!

4. Scalar variables

Python has two types of scalar values: numbers and strings. Both types ca be assigned to a scalar variable.

4.1 Numbers

Numbers are specified in any of the common integer or floating point format:

```
>>>x = 1                  # Integer
>>>y = 5.14               # Floating point
>>>z = 3.25E-7            # Scientific notation
```

Numbers can also be represented using binary or hexadecimal notations, but we will not need that.

Table of the most common number operators in Python:

Operator	**Meaning**
=	Assign
+	Add
-	Subtract
*	Multiply
/	Divide
**	Exponentiation
%	Modulus
abs(x)	Absolute value of x
int(x)	x converted to integer
float(x)	x converted to float
+=	Assign add
-=	Assign subtract
*=	Assign multiply
/=	Assign divide

Python allows us to use all standard arithmetic operators on numbers, plus a few others. The mathematical operations are performed in the standard order of precedence: power comes first, then multiplication has a higher precedence than addition and subtraction: 2+3*4 is equal to 14, and not 20. If we want the multiplication to be performed on 2+3, we need to include parentheses: (2+3)*4. These are exactly the rules used by Python.

Some of the operators listed in the table above are unusual, and require more explanations:

The modulo operator:

```
i=52
j=3
k=i%j
```

In the example given above, the variable k holds the remainder of the division of 52 by 3, i.e. 1.

Operating and assigning at once:

Operations that fetch a value from memory, modify it and store it back in memory are very common: Python has introduced a special syntax for those. Generally:

i = i <operator> b;

can be written as:

i <some operator> = b;

Let us see an example:

```
#
a = 5*4
print “5 times four is “, a, “\n”
$a +=4
print “Plus four is “,a,”\n”
$a/=3
print “Divided by three is “,a,”\n”
```

In this example, “a” takes successively the values 20, 24 and 8.

This works for +=, -=, *=, /=, **= and %=.

4.2 Strings

A string is a group of characters attached together, enclosed by quotation marks. For now, we will only consider double quotes.

Just like with numbers, many operations can be performed on strings: the most common ones are listed in the table below.

String operator	Meaning
a+b	Concatenates strings a and b
a*i	Repeats string a i times
a[i:j:k]	Returns a string containing all characters of a between position i and j, with step k; if k is negative, starts from the right
a[::-1]	Returns a string that is the reverse of a
a.split(sep)	Split string a into a list, using sep to decide where to cut
a.strip()	Returns a string equal to a, but that has been stripped of any "white" characters at the beginning and end of a (space, tab, CR,…)
a.upper()	Returns a string equal to a, but with all letters uppercase
a.lower()	Returns a string equal to a, but with all letters lowercase
a.capitalize()	Returns a string equal to a, but with the first word capitalized
a.count('sub')	Counts the number of instances of the substring 'sub' in the string a
a.replace('sub1','sub2',n)	Returns a string equal to a, but with n instances of substring sub1 replaced with substring sub2; if n is not given, all instances are returned

***Concatenating strings**:*

The + operator, when placed between two strings, creates a single string that concatenates the two original strings. In the following example:

```
#
>>>A==”ATTGC”
>>>B=”GGCCT”
>>>C=A+B
```

The variable C contains the string "ATTGCGGCCT". Note that the concatenation operator can be attached to an assignment:

C+=”G”;

Adds a "G" at the end of the string contained in C.

Repeating a string

The operator "*" repeats a string a given number of times:

```
>>> text=”No! “
>>>newtext=text*5
>>> print newtext
No! No! No! No! No!
```

Indexing and slicing strings

Characters within a string can be accessed both front and backwards. Frontways, a string starts at position 0 and the character desired is found via an offset value:
String[i] is the character at position **i** (starting from 0) from the **left** side of the string.

You can also find the same character by using a negative offset value from the end of the string:
String[-i] is the character at position **i** from the **right** side of the string.

```
>>> S = ‘Index’
>>> S[0]
I
>>> S[3]
e
>>> S[-1]
x
>>> S[-3]
d
```

Slicing is a very useful and heavily used function in Python: it allows you to extract specific substrings of a string. The syntax for slicing is:

b = S[i:j:k]

b collects characters between positions i and j (j not included), starting at I, every k characters.

Note that you do not have to specify i, j and/or k:

- if you do not specify i, you start at the first character of the string
- if you do not specify j, you go up to the last character of the string
- if you do not specify k, it is set by default to 1

Note also that k can be negative, in which case you start from the right end of the string. For example,

b = S[::-1]

reverses the string S and stores it in b.

Examples:

```
>>> S = 'This is a string'
>>> b = S[1:3]                  # Select substring from position 1 to 3, 3 not included
>>> print b
'hi'
>>> S[5:12:3]                   # Select every third character, between position 5 and 10
'iat'
>>> S[1:5:-1]                   # Starts from the end of the string; but  order 1:5 is wrong
                                  get nothing:
''
>>> S[5:1:-1]                   # correct syntax
'i si'
>>> S[10::]                     # all characters from position 10 till the end
'string'
>>> S[::-1]                     # reverse the whole string
'gnirts a si sihT'
```

The other string manipulations described below apply a function on the string. The syntax is:

string.function(argument)

where string is the string considered, function is the function applied, and argument are parameters for the function, if any.

Breaking a string into a list

A string can be broken down into a list using the function split. The syntax is:

A.split(sep)

where A is the string, and sep the separator. If sep is not provided, Python uses the white space.

Examples:

```
>>>text="This is a test case; it has two parts"
>>>text.split()
['This','is','a','test','case;','it','has','two','parts']
>>> text.split(';')
['This is a test case',' it has two parts']
>>> text.split('a')
['This is ',' test c','se; it h','s two p','rts']
```

Striping a string

A string may have leading or lagging white characters, such as blanks, tabs, or carriage return. It is a good idea to remove those, using the function strip().

Changing case

- Setting the whole string as upper case: apply function upper()
- Setting the whole string as lower case: apply function lower()
- Capitalizing the string: apply function capitalize()

```
>>> S = 'This Is A Test'
>>> S.upper()                          # All upper case
'THIS IS A TEST'
>>> S.lower()                          # All lower case
'this is a test'
>>> S.lower().capitalize()             # Set proper case
'This is a test'
>>> S = '       This is a test    '    # Remove leading and lagging tabs
'This is a test'
```

Counting occurrence of substrings

Count is a function that finds and counts the number of occurrence of a substring in a string:

```
>>> S='aattggccttaa'
>>> S.count('a')                       # Number of character 'a' in the string
4
>>> S.count('A')
0                                      # Remember that python is case sensitive
>>> S.count('at')                      # Number of 'at' in the string
1
>>>  S.count('Gc')
0
```

Replace

Replace is a function that substitutes a string for another:

String.replace('sub1','sub2',n)

String is the string on which replace is applied; n instances of 'sub1' are replaced with 'sub2'; if n is not provided, all instances of 'sub1' are replaced.

```
>>> S='This is a test case'
>>> S.replace('is','was')          # replaces all instances of 'is'
'Thwas was a test case'
>>> S.replace('is','was',1)        # replaces only first instance
'Thwas is a test case'
```

5. Input data in a Python program

Often when we write a Python script, we need to be able to ask the user for additional data when he/she runs the program. This is done using the function raw_input:

answer = raw_input("Question :")

where:

- "Question" is the string printed on the screen to let the user know what he/she needs to input
- answer is a string that stores the answer of the user.

Note that the result of raw_input is always a string. If you expect an integer or a float from the user, you need to change the type:

```
age = int(raw_input("What is your age :"))
```

age is now an integer that contains the age entered by the user.

Exercises:

1. Without the aid of a computer, work out the order in which each of the following expressions would be computed and their value.
   i. 2 + 6/4-3*5+1
   ii. 17 + -3**3/2
   iii. 26+3**4*2
   iv. 2*2**2+2

   Verify your answer using Python.

2. Without the aid of a computer, work out these successive expressions and give the values of a, b, c and d upon completion. Then check your answer using a Python script:

```
a=4
b=9
c=5
d= a*2+b*3
$c+=-$d/3
b%=a
a=b-1;
```

3. Write a Python program that:
   i. Reads a sentence from standard input
   ii. Writes this sentence on standard output all in lower case
   iii. Writes this sentence on standard output with all vowels in upper case and all consonants in lower case
   iv. Writes the sentence in reverse order

4. Write a Python program that:
   i. Reads a sentence from standard input
   ii. Counts the number of words and the number of characters, not included space
   iii. Counts the number of vowels.

5. Write a Python program that reads from standard input the amount of a restaurant bill and outputs two options for tipping, one based on 15% of the bill, the other based on 20% of the bill.

6. Write a Python program that:
   i. Reads a sentence
   ii. Remove all vowels
   iii. Replaces all v and b in the original sentence with b and v, respectively (i.e. for example string ‘cvvbt’ becomes ‘cbbvt’
   iv. Count number of letters in the modified sentence
   v. Writes the resulting sentence and number of letters on standard output