PYTHON BASICS

Mohan M J

Numbers and Expressions

Variables

Assignment

Statements

Getting Input from the User

Type - int and float

```
>>> 2 + 2
>>> 1 / 2  #division always returns a floating point number
>>> 1 // 2 # floor division discards the fractional part
>>> 1 % 2 # the % operator returns the remainder of the
division
>>> 10 / 3
>>> 10 % 3
>>> 2.75 % 0.5
>>> 2 ** 3  #use ** operator to calculate powers
>>> -3 ** 2
>>> (-3) ** 2
>>> width = 20   #equal sign (=) is used to assign a value
to a variable
>>> height = 5 * 9
>>> width * height                       900
>>>age = input("Enter age : ")
```

Functions, Modules

Built-in functions

Modules are extensions that can be imported into Python to extend its capabilities

cmath and Complex Numbers

```
>>> pow(2,3)                     8
>>> 10 + pow(2, 3*5)/3.0         10932.666666666666
>>> abs(-10)                     10
>>> ½                            0.5
>>> round(1.0/2.0)               0.0
>>> import math
>>> math.floor(32.9)             32.0
>>> int(32.0)                    32
>>> from math import sqrt
>>> sqrt(9)                      3.0
>>> sqrt(-1)                     nan
>>> import cmath
>>> cmath.sqrt(-1)               1j
>>> (1+3j) * (9+4j)              (-3+31j)
```

Strings

Concatenating Strings

Long Strings

Use '...' or "..." with the same result

Backslashes as escape quotes

```
>>> "Hello, world!"
>>> 'Hello, world!'
>>> 'Let's go!'       SyntaxError: invalid syntax
>>> 'Let\'s go!'      #use \ to escape the quotes in
                      the string
>>> x = "Hello, "
>>> y = "world!"
>>> x + y             'Hello, world!'
>>>print ('''This is a very long string.
       It continues here.
       "Hello, world!"
       Still here.''')
>>> path = 'C:\nowhere'  #here \n means newline!
>>> print (path)
>>> print ('C:\\nowhere')
>>> print (r'C:\nowhere') #note the r before the
                             quote
```

## Strings

```
>>> 'spam eggs'  # single quotes
'spam eggs'
>>> 'doesn\'t'  # use \' to escape
the single quote...
"doesn't"
>>> "doesn't"  # ...or use double
quotes instead
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
>>> "\"Yes,\" he said."
'"Yes," he said.'
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'
```

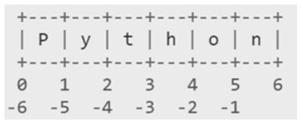
```
>>> '"Isn\'t," she said.'
'"Isn\'t," she said.'

>>> print('"Isn\'t," she said.')
"Isn't," she said.

>>> s = 'First line.\nSecond line.'
              # \n means newline
>>> s  # without print(), \n is included in the
              output
'First line.\nSecond line.'

>>> print(s)  # with print(), \n produces a new
line
First line.
Second line.
```

## Strings



```
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

Indexing

#Strings can be indexed (subscripted), with the first character having index 0. There is no separate character type; a character is simply a string of size one

#*slicing* is also supported. While indexing is used to obtain individual characters, *slicing* allows you to obtain substring

```
>>> word = 'Python'
>>> word[0]  # character in position 0      'P'
>>> word[5]  # character in position 5      'n'

#Indices may also be negative numbers, to start
counting from the right:
>>> word[-1]  # last character              'n'
>>> word[-2]  # second-last character       'o'
>>> word[-6]                                'P'

>>> word[0:2]  # characters from position 0
(included) to 2 (excluded)                  'Py'
>>> word[2:5]  # characters from position 2
(included) to 5 (excluded)                  'tho'

# s[:i] + s[i:] is always equal to s
>>> word[:2] + word[2:]              'Python'
>>> word[:4] + word[4:]              'Python'
```

Strings

#Python strings cannot be changed . They are immutable. Therefore, assigning to an indexed position in the string results in an error

```
>>> word[0] = 'J'
  ...
TypeError: 'str' object does not support item
assignment
>>> word[2:] = 'py'
  ...
TypeError: 'str' object does not support item
assignment

#If you need a different string, you should create a
new one
>>> 'J' + word[1:]                     'Jython'
>>> word[:2] + 'py'                    'Pypy'

#The built-in function len() returns the length of a
string:
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)                             34
```

Strings

#Python strings cannot be changed . They are immutable. Therefore, assigning to an indexed position in the string results in an error

```
>>> word[0] = 'J'
  ...
TypeError: 'str' object does not support item
assignment
>>> word[2:] = 'py'
  ...
TypeError: 'str' object does not support item
assignment

#If you need a different string, you should create a
new one
>>> 'J' + word[1:]                     'Jython'
>>> word[:2] + 'py'                    'Pypy'

#The built-in function len() returns the length of a
string:
>>> s = 'supercalifragilisticexpialidocious'
>>> len(s)                             34
```

Strings

```
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a string literal
  ...
SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
  ...
SyntaxError: invalid syntax
>>> prefix + 'thon'
'Python'
```

```
#String literals can span multiple lines
#End of lines are automatically included in the string
#but it's possible to prevent this by adding a \ at the end of the line
print("""\
Usage: thingy [OPTIONS]
     -h         Display this usage message
     -H hostname       Hostname to connect to
""")
#Strings can be concatenated (glued together) with the + operator, and repeated with *
>>> # 3 times 'un', followed by 'ium'
>>> 3 * 'un' + 'ium'
'unununium'
```

Strings

Item or slice assignments are illegal

String formatting operator (%)

Conversion specifiers

```
>>> website = 'http://www.python.org'
>>> website[-3:] = 'com'            :TypeError
#All kinds of item or slice assignments are illegal for strings
>>> format = "Hello, %s. %s enough for ya?"
>>> values = ('world', 'Hot')
>>> print (format % values)
Out [ ]: Hello, world. Hot enough for ya?

>>> format = "Pi with three decimals: %.3f"
>>> from math import pi
>>> print (format % pi)
              Pi with three decimals: 3.142
```

Summary

Expressions

Variables

Statements

Functions

Modules

Strings

abs(number)	Returns the absolute value of a number
cmath.sqrt(number)	Returns the square root; works with negative numbers
float(object)	Converts a string or number to a floating-point number
help()	Offers interactive help
input(prompt)	Gets input from the user
int(object)	Converts a string or number to an integer
long(object)	Converts a string or number to a long integer
math.ceil(number)	Returns the ceiling of a number as a float
math.floor(number)	Returns the floor of a number as a float
math.sqrt(number)	Returns the square root; doesn't work with negative numbers
pow(x, y)	Returns x to the power of y
input(prompt)	Gets input from the user, as a string
repr(object)	Returns a string representation of a value
round(number, ndigits)	Rounds a number to a given precision
str(object)	Converts a value to a string

List

Python knows a number of compound data types, used to group together other values. The most versatile is the list, which can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares                               [1, 4, 9, 16, 25]
#lists can be indexed and sliced
>>> squares[0]  # indexing returns the item          1
>>> squares[-1]                           25
>>> squares[-3:]  # slicing returns a new list
                                          [9, 16, 25]
#All slice operations return a new list containing the
requested elements
#This means that the following slice returns a new copy
of the list
>>> squares[:]                            [1, 4, 9, 16, 25]
>>> squares + [36, 49, 64, 81, 100]    #concatenation
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Lists

Indexing

Slicing

Appending

Unlike strings, which are immutable, lists are a mutable type. It is possible to change their content

```
>>> cubes = [1, 8, 27, 65, 125]  # something's wrong here
>>> 4 ** 3  # the cube of 4 is 64, not 65!
                             64
>>> cubes[3] = 64  # replace the wrong value
>>> cubes             [1, 8, 27, 64, 125]
#Add new items at the end of the list, by using the append() method
>>> cubes.append(216)  # add the cube of 6
>>> cubes.append(7 ** 3)  # and the cube of 7
>>> cubes
[1, 8, 27, 64, 125, 216, 343]
```

Lists

Assignment to slices is possible
This can even change the size of the list or clear it entirely

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
# replace some values
>>> letters[2:5] = ['C', 'D', 'E']
>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
# remove thems
>>> letters[2:5] = []
>>> letters                      ['a', 'b', 'f', 'g']
# clear the list by replacing all the elements with an empty list
>>> letters[:] = []
>>> letters                      []
#The built-in function len() also applies to lists
>>> letters = ['a', 'b', 'c', 'd']
>>> len(letters)                 4
```

Lists

Sequence Overview

Common Sequence Operations

indexing, *slicing*, *adding*, *multiplying*, and checking for *membership*

```
>>> edward = ['Edward Gumby', 42]
>>> john = ['John Smith', 50]
>>> database = [edward, john]
>>> database
                [['Edward Gumby', 42], ['John Smith',
50]]
>>> greeting = 'Hello'
>>> greeting[0]          'H'
>>> greeting[-1]         'o'
#It is possible to nest lists (create lists containing
other lists)
>>> a = ['a', 'b', 'c']
>>> n = [1, 2, 3]
>>> x = [a, n]
>>> x                   [['a', 'b', 'c'], [1, 2, 3]]
>>> x[0]                ['a', 'b', 'c']
>>> x[0][1]             'b'
```

Lists

Indexing Example

Print out a date given year, month, and day as numbers

Output Eg: August 16th, 1974

Exercise: Extend this program for the input in DD/MM/YY format

```
# Print out a date given year, month, and day as numbers
# (Out Eg: August 16th, 1974)
months = [ 'January', 'February', 'March', 'April', 'May',
'June', 'July', 'August', 'September', 'October', 'November',
'December']
# A list with one ending for each number from 1 to 31
endings = ['st', 'nd', 'rd'] + 17 * ['th'] \
         + ['st', 'nd', 'rd'] + 7 * ['th'] + ['st']
day      = input('Day (1-31): ')
month    = input('Month (1-12): ')
year     = input('Year: ')
month_number      = int(month)
day_number        = int(day)
# Remember to subtract 1 from month and day to get a correct
index
month_name        = months[month_number-1]
ordinal           = day + endings[day_number-1]
print (month_name + ' ' + ordinal + ', ' + year)
```

Slicing

```
>>> numbers[0:10:1]
         [1, 2, 3, 4, 5, 6, 7, 8, 9 , 10]
>>> numbers[0:10:2]          #longer steps
         [1, 3, 5, 7, 9]
>>>numbers[3:6:3]            [4]
>>> numbers[::4]             [1, 5, 9]
>>> numbers[8:3:-1]          [9, 8, 7, 6, 5]
>>> numbers[10:0:-2]         [10, 8, 6, 4, 2]
>>> numbers[0:10:-2]         []
>>> numbers[::-2]            [10, 8, 6, 4, 2]
>>> numbers[5::-2]           [6, 4, 2]
>>> numbers[:5:-2]           [10, 8]
```

```
>>> tag = '<a href="http://www.python.org">Python web
site</a>'
>>> tag[9:30]               'http://www.python.org'
>>> tag[32:-4]              'Python web site'
>>> numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> numbers[3:6]            [4, 5, 6]
>>> numbers[0:1]            [1]
>>> numbers[7:10]           [8, 9, 10]
>>> numbers[-3:-1]          [8, 9]
>>> numbers[-3:0]           []
>>> numbers[-3:]            [8, 9, 10]
>>> numbers[:3]             [1, 2, 3]
>>> numbers[:]              [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Split up a URL of the form http://www.something.com
url = input('Please enter the URL: ')
domain = url[11:-4]
print ("Domain name: " + domain)
```

Sequences

Adding

Multiplication

Membership

```
>>> [1, 2, 3] + [4, 5, 6]       [1, 2, 3, 4, 5, 6]
>>> 'Hello, ' + 'world!'        'Hello, world!'
>>> [1, 2, 3] + 'world!'        ? Error!
>>> 'python' * 5
                'pythonpythonpythonpythonpython'
>>> [42] * 10
[42, 42, 42, 42, 42, 42, 42, 42, 42, 42]

>>> permissions = 'rw'
>>> 'w' in permissions          True
>>> 'x' in permissions          False
>>> users = ['mlh', 'foo', 'bar']
>>>input('Enter your user name: ') in users
>>> subject = '$$$ Get rich now!!! $$$'
>>> '$$$' in subject                    True
```

Sequence

Membership Example

```
# Check a user name and PIN code
database = [ \
       ['albert', '1234'], \
       ['dilbert', '4242'], \
       ['smith', '7524'], \
       ['jones', '9843'] \
       ]
username     = input('User name: ')
pin          = input('PIN code: ')
if [username, pin] in database: print(
'Access granted')
```

Lists

Length

Minimum

Maximum

```
>>> numbers = [100, 34, 678]
>>> len(numbers)          3
>>> max(numbers)          678
>>> min(numbers)          34
>>> max(2, 3)             3
>>> min(9, 3, 2, 5)       2
```

Lists

Basic List Operations

Changing Lists: Item Assignments

Deleting Elements

Assigning to Slices

```
>>> list('Hello')         ['H', 'e', 'l', 'l', 'o']
>>> x = [1, 1, 1]
>>> x[1] = 2
>>> x                     [1, 2, 1]

>>> names = ['Alice', 'Beth', 'Cecil', 'Dee-Dee', 'Earl']
>>> del names[2]
>>> names                 ['Alice', 'Beth', 'Dee-Dee', 'Earl']
>>> name = list('Perl')
>>> name                  ['P', 'e', 'r', 'l']
>>> name[2:] = list('ar')
>>> name                  ['P', 'e', 'a', 'r']
>>> name = list('Perl')
>>> name[1:] = list('ython')
>>> name                  ['P', 'y', 't', 'h', 'o', 'n']

>>> numbers = [1, 5]
>>> numbers[1:1] = [2, 3, 4]
>>> numbers                          [1, 2, 3, 4, 5]
>>> numbers                          [1, 2, 3, 4, 5]
>>> numbers[1:4] = []
>>> numbers                          [1, 5]
```

List Methods

object.method(arguments)

Append

Extend

```
>>> lst = [1, 2, 3]
>>> lst.append(4)
>>> lst                              [1, 2, 3, 4]
>>> ['to', 'be', 'or', 'not', 'to', 'be'].count('to')
        2
>>> x = [[1, 2], 1, 1, [2, 1, [1, 2]]]
>>> x.count(1)                       2
>>> x.count([1, 2])                  1
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> a.extend(b)
>>> a                                [1, 2, 3, 4, 5, 6]
>>> a = [1, 2, 3]
>>> a + b                            [1, 2, 3, 4, 5, 6]
>>> a                                [1, 2, 3]
>>> a[len(a):] = b
>>> a                                [1, 2, 3, 4, 5, 6]
```

List Methods

Index

Insert

Pop

```
>>> knights = ['We', 'are', 'the', 'knights', 'who', 'say', 'ni']
>>> knights.index('who')           4
>>> knights.index('herring')                Value Error
#Insert
>>> numbers = [1, 2, 3, 5, 6, 7]
>>> numbers.insert(3, 'four')
>>> numbers                        [1, 2, 3, 'four', 5, 6, 7]
>>> numbers = [1, 2, 3, 5, 6, 7]
>>> numbers[3:3] = ['four']
>>> numbers                        [1, 2, 3, 'four', 5, 6, 7]
#Pop
>>> x = [1, 2, 3]
>>> x.pop()                        3
>>> x                              [1, 2]
>>> x.pop(0)                       1
>>> x                              [2]
```

List Methods

Remove

Reverse

Sort

```
>>> x = ['to', 'be', 'or', 'not', 'to', 'be']
>>> x.remove('be')
>>> x           ['to', 'or', 'not', 'to', 'be']
>>> x.remove('bee')             ValueError
>>> x = [1, 2, 3]
>>> x.reverse()
>>> x                           [3, 2, 1]
>>> x = [4, 6, 2, 1, 7, 9]
>>> x.sort()
>>> x                           [1, 2, 4, 6, 7, 9]
>>> x = [4, 6, 2, 1, 7, 9]
>>> y = x.sort() # Don't do this!
>>> print (y)                   None
>>> y = x[:]
>>> y.sort()
>>> x                           [4, 6, 2, 1, 7, 9]
>>> y                           [1, 2, 4, 6, 7, 9]
>>> y = x   #Dont do this!
>>> y.sort()
>>> x                           [1, 2, 4, 6, 7, 9]
>>> y                           [1, 2, 4, 6, 7, 9]
```

Tuples:

Immutable Sequences

The tuple Function

Basic Tuple Operations

Separate some values with commas, you automatically have a tuple

```
>>> 1, 2, 3                     (1, 2, 3)
>>> (1, 2, 3)                   (1, 2, 3)
>>> () #empty tuple             ()
#tuple containing a single value
>>> 42                          42
>>> 42,                         (42,)
>>> (42,)                       (42,)
>>> 3*(40+2)                    126
>>> 3*(40+2,)                   (42, 42, 42)
>>> tuple([1, 2, 3])            (1, 2, 3)
>>> tuple('abc')                ('a', 'b', 'c')
>>> tuple((1, 2, 3))            (1, 2, 3)
>>> x = 1, 2, 3
>>> x[1]                        2
>>> x[0:2]                      (1, 2)
```

Summary

Sequences

Membership

Methods

`len(seq)`	Returns the length of a sequence
`list(seq)`	Converts a sequence to a list
`max(args)`	Returns the maximum of a sequence or set of arguments
`min(args)`	Returns the minimum of a sequence or set of arguments
`sorted(seq)`	Returns a sorted list of the elements of seq
`tuple(seq)`	Converts a sequence to a tuple

THANK YOU