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

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

Raw strings

```
>>> "Hello, world!"
>>> 'Hello, world!'
>>> 'Let's go!'       SyntaxError: invalid syntax
>>> 'Let\'s go!'      #use \ to escape 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') #But for long paths!!
>>> print (r'C:\nowhere') #note the r before the quote
```

Strings

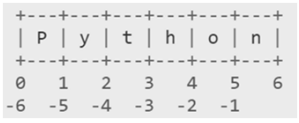
```
>>> 'spam eggs'           # single quotes
>>> 'doesn\'t'            # use \' to escape the single quote
>>> "doesn't"             # ...or use double quotes instead
>>> '"Yes," they said.'
>>> "\"Yes,\" they said.“

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

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

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

Strings



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'
        Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
        TypeError: 'str' object does not support item assignment
>>> word[2:] = 'py'
        Traceback (most recent call last):
        File "<stdin>", line 1, in <module>
        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

```
#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 * 'un' + 'ium'         # 3 times 'un', followed by 'ium‘
>>> 'Py' 'thon‘              #Two or more string literals next to each other are
                             automatically concatenated
```

Strings

```
>>> prefix = 'Py'
>>> prefix 'thon'  # can't concatenate a variable and a string literal
               File "<stdin>", line 1
               prefix 'thon'
               SyntaxError: invalid syntax
>>> ('un' * 3) 'ium'
               File "<stdin>", line 1
               ('un' * 3) 'ium'
               SyntaxError: invalid syntax
>>> prefix + 'thon'          'Python'
```

String Formatting

You can use %s to inject strings into your print statements.

The modulo % is referred to as a "string formatting operator".

```
>>> format = "Hello, %s. %s enough for ya?"
>>> values = ('world', 'Hot')
>>> print (format % values)
                Out [ ]: Hello, world. Hot enough for ya?
```

```
#Padding and Precision of Floating Point Numbers
>>> print('Floating point numbers: %5.2f' %(13.144))
>>> print('Floating point numbers: %25.2f' %(13.144))
>>> format = "Pi with three decimals: %.3f"
>>> from math import pi
>>> print (format % pi)
>>> print('First: %s, Second: %5.2f, Third: %r'
       %('hi!',3.1415,'bye!'))
```

String Formatting

Formatting with the .format() method

```
>>> print('This is a string with an {}'.format('insert'))

# Inserted objects can be called by index position
>>> print('The {2} {1} {0}'.format('fox','brown','quick'))
>>> "{3} {0} {2} {1} {3} {0}".format("be", "not", "or", "to")
                                        'to be or not to be'
#Inserted objects can be assigned keywords
>>> print('First Object: {a}, Second Object: {b}, Third
       Object: {c}'.format(a=1,b='Two',c=12.3))

#Inserted objects can be reused, avoiding duplication:
>>> print('A {p} saved is a {p} earned.'.format(p='penny'))
```

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]
>>> print(squares)
#lists can be indexed and sliced
>>> squares[0]  # indexing returns the item
>>> squares[-1]
>>> squares[-3:]  # slicing returns a new list
#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
```

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!
>>> 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*

```
#It is possible to nest lists (create lists containing other
lists)
>>> edward = ['Edward Gumby', 42]
>>> john = ['John Smith', 50]
>>> database = [edward, john]
>>> database         [['Edward Gumby', 42], ['John Smith', 50]]

>>> 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

```
months = [ 'January', 'February', 'March', 'April', 'May',
'June', 'July', 'August', 'September', 'October', 'November',
'December']
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]   [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]
```

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

Lists

Basic List Operations

Changing Lists: Item Assignments

Deleting Elements

Assigning to Slices

```
>>> 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')
>>> 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

clear

copy

```
>>> lst = [1, 2, 3]
>>> lst.clear()
>>> lst                                  []
#normal assignment simply binds another name to the same
list
>>> a = [1, 2, 3]
>>> b = a
>>> b[1] = 4
>>> a                                    [1, 4, 3]
If you want a and b to be separate lists, you have to
bind b to a copy of a
>>> a = [1, 2, 3]
>>> b = a.copy()
>>> b[1] = 4
>>> a                                    [1, 2, 3]
```

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 = x.copy()
>>> 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]
```

List Methods

reversed

sorted

```
>>> x = [1, 2, 3]
>>> list(reversed(x))                [3, 2, 1]
>>> x = [4, 6, 2, 1, 7, 9]
>>> y = sorted(x)
>>> x                                [4, 6, 2, 1, 7, 9]
>>> y                                [1, 2, 4, 6, 7, 9]
>>> sorted('Python')         ['P', 'h', 'n', 'o', 't', 'y']

#if you want to sort the elements according to their lengths,
you use len as the key function
>>> x = ['aardvark', 'abalone', 'acme', 'add', 'aerate']
>>> x.sort(key=len)
>>> x          ['add', 'acme', 'aerate', 'abalone', 'aardvark']

# reverse sorting
>>> x = [4, 6, 2, 1, 7, 9]
>>> x.sort(reverse=True)
>>> x                                [9, 7, 6, 4, 2, 1]
```

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

String Methods

find

```
#returns the leftmost index where the substring is
found. If it is not found, –1 is returned
>>> 'With a moo-moo here, and a moo-moo there'.find('moo')
                                                        7
>>> title = "Monty Python's Flying Circus"
>>> title.find('Monty')                                 0
>>> title.find('Python')                                6
>>> title.find('Flying')                                15
>>> title.find('Zirquss')                               -1
>>> subject = '$$$ Get rich now!!! $$$'
>>> subject.find('$$$')                                 0
>>> subject.find('$$$', 1) # Only supplying the start
                                                        20
```

String Methods

join

```
#join is the inverse of split.
#It is used to join the elements of a sequence
>>> seq = [1, 2, 3, 4, 5]
>>> sep = '+'
>>> sep.join(seq) # Trying to join a list of numbers
                Traceback (most recent call last):
                File "<stdin>", line 1, in ?
        TypeError: sequence item 0: expected string, int found
>>> seq = ['1', '2', '3', '4', '5']
# Joining a list of strings
>>> sep.join(seq)                          '1+2+3+4+5'
>>> dirs = '', 'usr', 'bin', 'env'
>>> '/'.join(dirs)                         '/usr/bin/env'
>>> print('C:' + '\\'.join(dirs))          C:\usr\bin\env
```

String Methods

Lower

Replace

Split

```
#The lower method returns a lowercase version of the string
>>> 'Trondheim Hammer Dance'.lower()
>>> name = 'Gumby'
>>> names = ['gumby', 'smith', 'jones']
>>> if name.lower() in names: print('Found it!')

#replace method returns a string where all the occurrences of one
string have been replaced by another
>>> 'This is a test'.replace('is', 'eez')          'Theez eez a test'

#split used to split a string into a sequence.
>>> '1+2+3+4+5'.split('+')          ['1', '2', '3', '4', '5']
>>> '/usr/bin/env'.split('/')       ['', 'usr', 'bin', 'env']
>>> 'Using the default'.split()     ['Using', 'the', 'default']
```

String Methods

Strip

```
#strip method returns a string where whitespace on the left and right (but not internally) has been stripped (removed)
>>> ' internal whitespace is kept '.strip()
>>> names = ['gumby', 'smith', 'jones']
>>> name = 'gumby '
>>> if name in names: print('Found it!')
>>> if name.strip() in names: print('Found it!')
                                                    Found it!
>>> '*** SPAM * for * everyone!!! ***'.strip(' *!')
                                                    'SPAM * for * everyone'
```

String Methods

Translate

```
# make a translation table
>>> table = str.maketrans('cs', 'kz')
>>> 'this is an incredible test'.translate(table)
'thiz iz an inkredible tezt'

# An optional third argument can be supplied to maketrans, specifying letters that should be deleted
>>> table = str.maketrans('cs', 'kz', ' ')
>>> 'this is an incredible test'.translate(table)
'thizizaninkredibletezt'
```

THANK YOU