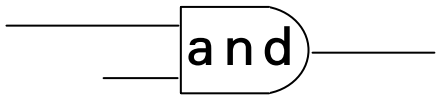
Mathematics
for the Digital Age



Programming
in Python

>>> Second Edition:
with Python 3

Maria Litvin
Phillips Academy, Andover, Massachusetts

Gary Litvin
Skylight Software, Inc.

Skylight Publishing
Andover, Massachusetts

Skylight Publishing
9 Bartlet Street, Suite 70
Andover, MA 01810

web: http://www.skylit.com
e-mail: sales@skylit.com
support@skylit.com

Library of Congress Control Number: 2009913596

ISBN 978-0-9824775-8-8

1 2 3 4 5 6 7 8 9 10 15 14 13 12 11 10

Printed in the United States of America

Appendix A. Getting Started with Python

This appendix is on the Internet at:

`www.skylit.com/python`

Appendix B. Selected Functions: Built-In, `Math`, and `Random`

`help(obj)`	displays help for a function or module, as in
	`>>> import math`
	`>>> help(math)`

`input(s)`	displays `s` as a prompt, then reads a line of text, typed in by the user, and returns it as a string
`abs(x)`	returns the absolute value of `x`
`max(a, b)`	returns the largest of `a`, `b`
`min(a, b)`	returns the smallest of `a`, `b`
`int(s)`	converts a string or a `float` into an integer
`float(s)`	converts a string or an `int` into a `float`
`str(n)`	converts `n` into a string
`bin(n)`	returns a string that represents `n` in binary
`hex(n)`	returns a string that represents `n` in hex
`oct(n)`	returns a string that represents `n` in octal
`len(s)`	returns the length of a string, list, or tuple
`sum(lst)`	returns the sum of the numbers from a list or tuple
`max(lst)`	returns the largest element of a list or tuple
`min(lst)`	returns the smallest element of a list or tuple
`list(s)`	converts a string or tuple into a list
`tuple(s)`	converts a string or list into a tuple
`range(n)`	generates `0, ..., n-1`, as in: `for i in range(n):`
`open(pathname)`	opens a file
`open(pathname, 'w')`	creates a file and opens it for writing

`from math import *`		`from random import *`	
`pi`	3.14159...	`x = random()`	$0 \le x < 1$
`e`	2.71828...	`r = randint(a, b)`	$a \le r \le b$
`sqrt(x)`	$\sqrt{x}$	`a = choice(s)`	`s[r]`
`pow(x, y)`	x^y	`shuffle(lst)`	shuffles `lst`
`exp(x)`	e^x		

Appendix C. String Operations and Methods

Contents type

The following methods return `True` if all the characters in `s` belong to the corresponding category; otherwise return `False`:	
`s.isalpha()`	Checks whether all the characters in `s` are letters
`s.isdigit()`	Checks whether all the characters in `s` are digits
`s.isalnum()`	Checks whether each character in `s` is either a letter or a digit
`s.isupper()`	Checks whether all the letters in `s` are upper case
`s.islower()`	Checks whether all the letters in `s` are lower case
`s.isspace()`	Checks whether all characters in `s` are "white space" (spaces, tabs, newline, etc.)

Examples:

```
>>> 'ab7'.isalpha()
False
>>> 'ab7'.isdigit()
False
>>> 'ab7'.isalnum()
True
>>> 'ab7'.isupper()
False
>>> 'ab7'.islower()
True
>>> ' * '.isspace()
False
>>> '  \n'.isspace()
True
```

Length and substrings

`len(s)`	Returns the number of characters in `s`
`ch = s[i]`	Sets `ch` to the *i*-th character in `s`
`s2 = s[i:j]`	Sets `s2` to the substring of `s` from `i` to `j-1`

Examples:

```
>>> len('abcd')
4
>>> 'abcd'[1]
'b'
```

```
>>> 'abcd'[1:3]
'bc'
>>> 'abcd'[:3]
'abc'
```

Search

The following methods return an `int`:	
`s.find(sub)`	Returns the index of the first occurrence of `sub` in `s`; if `sub` is not found, returns `-1`
`s.rfind(sub)`	Returns the index of the last occurrence of `sub` in `s`
`s.count(sub)`	Returns the number of times `sub` occurs in `s`

`s.find(sub, start, end)`
`s.rfind(sub, start, end)`
`s.count(sub, start, end)`
	The versions with optional arguments `start`, `end`, look for `sub` only within the substring of `s` between `start` and `end-1`

Examples:

```
>>> 'never'.find('e')
1
>>> 'never'.find('x')
-1
>>> 'never'.rfind('e')
3
>>> 'never'.count('e')
2
```

```
>>> 'never'.find('ver')
2
>>> 'never'.find('e',2,4)
3
>>> 'never'.rfind('e',1,3)
1
>>> 'never'.find('ver',2,4)
-1
```

Case conversions

The following methods return a new string:	
`s.upper()`	All the letters are converted to upper case
`s.lower()`	All the letters are converted to lower case
`s.capitalize()`	The first letter is converted to upper case

Examples:

```
>>> 'ab7'.upper()
'AB7'
>>> 'Ab7'.lower()
'ab7'
>>> 'ab7'.capitalize()
'Ab7'
>>> '7ab'.capitalize()
'7ab'
```

Editing

The following methods return a new string:	
`s.replace(old, new)`	Replaces every occurrence of `old` in `s` with `new`
`s.strip()`	Removes white space at the beginning and at the end of `s`

Examples:

```
>>> '1*2*3'.replace('*','--')
'1--2--3'
>>> ' ab   \n'.strip()
'ab'
```

Parsing

The following methods return a list:	
`s.split(delim)`	returns a list of substrings separated by occurrences of `delim` in `s`
`s.splitlines()`	returns a list of lines in `s` — the same as `s.split('\n')`.

Examples:

```
>>> 'Line1\nLine 2'.splitlines()
['Line1', 'Line 2']
```

```
>>> '1, 2, 3'.split(', ')
['1', '2', '3']
```

Formatting

The following methods return a new string:	
`s.format(value,...)`	Formats `value` (or several values according to the `format` fields in `s`
`s.ljust(w, fill)`	Left-justifies `s` within a string of length `w` and pads it on the right with the `fill` character (`fill` is optional: if not given, `ljust`, `rjust` and `center`, use the space character as the default)
`s.rjust(w, fill)`	Right-justifies `s` and pads it on left with `fill`
`s.center(w, fill)`	Positions `s` in the middle of a string of length `w` and pads it on both sides with `fill`
`s.zfill(w)`	Right-justifies the string and pads it with 0s on the left — the same as `s.rjust(w,'0')`

Examples:

```
>>> '{0:>4s}{1:7.2f}'.format('$',2.5)
'   $   2.50'
>>> 'ab'.ljust(6, '*')
'ab****'
```

```
>>> 'ab'.rjust(6)
'    ab'
>>> 'ab'.center(6)
'  ab  '
>>> '12'.zfill(4)
'0012'
```

Appendix D. List, Set, and Dictionary Operations and Methods

Lists

Method/Operation	Description
`len(lst)`	Returns the number of elements in `lst`
`x = lst[i]`	Sets `x` to the *i*-th element of `lst`
`lst[i] = x`	Sets the *i*-th element of `lst` to `x`
`del lst[i]`	Deletes the *i*-th element and decrements the indices of the subsequent elements by one
`del lst[i:j]`	Deletes the slice from *i* to *j* and adjusts the indices of the subsequent elements
`lst2 = lst[i:j]`	Creates a copy of the specified slice from `lst` and assigns it to `lst2`
`lst2 = lst[:]`	Creates a copy of `lst` and assigns it to `lst2`
`lst.insert(i, x)`	Inserts `x` at index `i`, shifting the subsequent elements to the right by 1
`lst.append(x)`	Appends `x` at the end of `lst`
`lst.pop(i)`	Returns the *i*-th element and removes it from `lst`
`lst.pop()`	Returns the last element and removes it from `lst`
`lst.remove(x)`	Removes the first occurrence of `x` from `lst`; raises an exception if none found
`lst.index(x)`	Returns the index of the first occurrence of `x` in `lst`; raises an exception if none found
`lst.count(x)`	Returns the number of times `x` occurs in `lst`
`lst.reverse()`	Reverses the order of elements in `lst`; returns `None`
`lst.sort()`	Arranges the elements of `lst` in ascending order; returns `None`

Examples:

```
>>> lst=['A', 'C']
>>> lst
['A', 'C']
>>> lst.insert(1, 'B')
>>> lst
['A', 'B', 'C']
>>> lst.append('A')
>>> lst
['A', 'B', 'C', 'A']
>>> lst.insert(2, 'A')
>>> lst
['A', 'B', 'A', 'C', 'A']
>>> lst.count('A')
3
>>> del lst[2]
>>> lst
['A', 'B', 'C', 'A']
>>> lst.reverse()
>>> lst
['A', 'C', 'B', 'A']
>>> lst.index('A')
0
>>> lst.remove('A')
>>> lst
['C', 'B', 'A']
>>> lst.sort()
>>> lst
['A', 'B', 'C']
>>> lst.pop(1)
'B'
>>> lst
['A', 'C']
>>> lst.pop()
'C'
>>> lst
['A']
```

Sets

Method/Operation	Description
`len(s)`	Returns the number of elements in `s`
`s.copy()`	Returns a copy of `s`
`s.add(x)`	Adds `x` to `s`
`s.remove(x)`	Removes `x` from `s`; raises an exception if `x` is not in `s`
`s.discard(x)`	Removes `x` from `s`; has no effect if `x` is not in `s`
`s.pop()`	Removes and returns an arbitrary element from `s`
`s1.issubset(s2)`	Returns `True` if `s1` is a subset of `s2`
`s.update(s2)`	Adds all the elements from a list, tuple, or set `s2` to `s`

Examples:

```
>>> s = {'A', 'C'}
>>> s
{'A', 'C'}
>>> s.add('B')
>>> s
{'A', 'C', 'B'}
>>> s.remove('A')
>>> s
{'C', 'B'}
>>> s.discard('X')
>>> s
{'C', 'B'}
>>> s.pop()
'C'
>>> s
{'B'}
>>> s2 = set('ABCD')
>>> s2
{'A', 'C', 'B', 'D'}
```

```
>>> s.issubset(s2)
True
>>> s.add('X')
>>> s
{'X', 'B'}
>>> s.issubset(s2)
False
>>> s.update(s2)
>>> s
{'A', 'C', 'B', 'D', 'X'}
```

Dictionaries

Method/Operation	Description
`len(d)`	Returns the number of key-value pairs in `d`
`x = d[k]`	Sets `x` to the value associated with the key `k` in `d`
`d[k] = x`	If the key `k` is in `d`, changes the value associated with `k` to `x`; if `k` is not in `d`, adds the `k:x` pair to `d`
`del d[k]`	Deletes the key `k` and the associated value from `d`
`d2 = d.copy()`	Creates a copy of `d` and assigns it to `d2`
`k in d`	Returns `True` if the key `k` is in `d`; otherwise returns `False`
`d.keys()`	Returns a `dict_keys` object that contains all the keys in `d`
`d.items()`	Returns a set (a `dict_items` object) of all (key, value) pairs in `d`
`d.update(d2)`	Adds all the key-value pairs from `d2` to `d`
`d.get(k)`	The same as `d[k]`
`d.get(k, dflt)`	If `k` is in `d`, returns `d[k]`; otherwise returns `dflt`

Examples:

```
>>> d = {'K1':'V1', 'K2':'V2'}
>>> d.keys()
dict_keys(['K2', 'K1'])
>>> d.items()
dict_items([('K2', 'V2'), ('K1', 'V1')])
>>> 'K2' in d
True
>>> del d['K2']
>>> d
{'K1': 'V1'}
>>> 'K2' in d
False
>>> d['K2'] = 'V2'
>>> d
{'K2': 'V2', 'K1': 'V1'}
```

```
>>> d.get('K2')
'V2'
>>> d.get('X', 'oops')
'oops'
```