List Methods

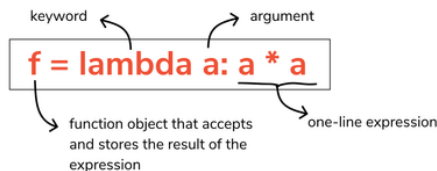| Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list (shallow copy)
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Operator Precedence and Associativity

Precedence	Operators	Description	Associativity
1	()	Parentheses	Left to right
2	x[index], x[index:index]	Subscription, slicing	Left to right
3	await x	Await expression	N/A
4	**	Exponentiation	Right to left
5	+x, -x, ~x	Positive, negative, bitwise NOT	Right to left
6	*, @, /, //, %	Multiplication, matrix, division, floor division, remainder	Left to right
7	+, -	Addition and subtraction	Left to right
8	<<, >>	Shifts	Left to right
9	&	Bitwise AND	Left to right
10	^	Bitwise XOR	Left to right
11			Bitwise OR	Left to right
12	in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, membership tests, identity tests	Left to Right
13	not x	Boolean NOT	Right to left
14	and	Boolean AND	Left to right
15	or	Boolean OR	Left to right
16	if-else	Conditional expression	Right to left
17	lambda	Lambda expression	N/A
18	:=	Assignment expression (walrus operator)	Right to left

Set Operations

Operation	Equivalent	Result
len(s)		number of elements in set s (cardinality)
x in s		test x for membership in s
x not in s		test x for non-membership in s
s.issubset(t)	s <= t	test whether every element in s is in t
s.issuperset(t)	s >= t	test whether every element in t is in s
s.union(t)	s	t	new set with elements from both s and t
s.intersection(t)	s & t	new set with elements common to s and t
s.difference(t)	s - t	new set with elements in s but not in t
s.symmetric_difference(t)	s ^ t	new set with elements in either s or t but not both
s.copy()		new set with a shallow copy of s

File Opening Modes

Modes	Description
r	Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
rb	Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
r+	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
rb+	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
w	Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
wb	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
w+	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
wb+	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
a	Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
ab	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
a+	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
ab+	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

keyword ← → argument

f = lambda a: a * a

function object that accepts and stores the result of the expression

one-line expression

Dictionary Methods

Method	Description
clear()	Removes all the elements from the dictionary
copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
update()	Updates the dictionary with the specified key-value pairs
values()	Returns a list of all the values in the dictionary

Tuple Methods

Method	Description
count()	Returns the number of times a specified value occurs in a tuple
index()	Searches the tuple for a specified value and returns the position of where it was found

File Methods

Method	Description
close()	Closes the file
detach()	Returns the separated raw stream from the buffer
fileno()	Returns a number that represents the stream, from the operating system's perspective
flush()	Flushes the internal buffer
isatty()	Returns whether the file stream is interactive or not
read()	Returns the file content
readable()	Returns whether the file stream can be read or not
readline()	Returns one line from the file
readlines()	Returns a list of lines from the file
seek()	Change the file position
seekable()	Returns whether the file allows us to change the file position
tell()	Returns the current file position
truncate()	Resizes the file to a specified size
writable()	Returns whether the file can be written to or not
write()	Writes the specified string to the file
writelines()	Writes a list of strings to the file

String Comparison

If you compare strings that contain the same substring, such as Apple and ApplePie, then the longer string is considered larger.

Python string comparison compares the characters in both strings one by one. When different characters are found, then their Unicode code point values are compared. The character with the lower Unicode value is considered to be smaller.

Important Things

In a dictionary in Python, values can be any data type, but keys cannot be list, dictionary, and set (unhashable type)

In string slicing, the format is [start : stop : step]. Stop is **always** exclusive.
If step is 1 (default value), the default value of start will be 0.
If step is set to be -1, the default value of start will change to -1.

ord('A') = 65   chr(65) = 'A'

In set operation, |, &, and ^ are associative. On the other words. ((set1 ^ set2) ^ set3) = (set1 ^ (set2 ^ set3))

Conversion must be primitive to primitive, or iterables to iterables

Map and Filter

Syntax: map(fun, iter)
Parameters:
- fun: It is a function to which map passes each element of given iterable.
- iter: iterable object to be mapped.

Note: map function is only shallow map

Syntax: filter(function, sequence)
Parameters:
- function: function that tests if each element of a sequence is true or not.
- sequence: sequence which needs to be filtered, it can be sets, lists, tuples, or containers of any iterators.

Note: map and filter will return an object. This object is consumable, so you can access them once. To make it permanent, you can change its type first by using list() or tuple().

Flatten

```
def flatten(seq):
    if seq == []:
        return seq
    if type(seq) != list:
        return [seq]
    return flatten(seq[0]) + flatten(seq[1:])
```

String Methods

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
expandtabs()	Sets the tab size of the string
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
format_map()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isascii()	Returns True if all characters in the string are ascii characters
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Converts the elements of an iterable into a string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
maketrans()	Returns a translation table to be used in translations
partition()	Returns a tuple where the string is parted into three parts
replace()	Returns a string where a specified value is replaced with a specified value
rfind()	Searches the string for a specified value and returns the last position of where it was found
rindex()	Searches the string for a specified value and returns the last position of where it was found
rjust()	Returns a right justified version of the string
rpartition()	Returns a tuple where the string is parted into three parts
rsplit()	Splits the string at the specified separator, and returns a list
rstrip()	Returns a right trim version of the string
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
strip()	Returns a trimmed version of the string
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Deep Map and Deep Copy

```
def deepMap(f, seq):
    if seq == []:
        return seq
    if type(seq) != list:
        return f(seq)
    return [deepMap(f, seq[0])] + deepMap(f, seq[1:])


def deepCopy(seq):
    return deepMap(lambda x:x, seq)
```

Written by Jansen Ken Pegrasio

List Methods Examples

```python
x = [1, 2]
x.append(3)
# x will become [1, 2, 3]
x.extend([4, 5])
# x will become [1, 2, 3, 4, 5]
x.pop(-1)
# pop the last element of x
# x will become [1, 2, 3, 4]
x.insert(3, 1)
# insert 1 at the 3rd index
# x will become [1, 2, 3, 1, 4]
a = x.copy()
# a is a shallow copy of x, which is [1, 2, 3, 1, 4]
b = x.count(1)
# count() will return the number of 1s in x, so b will be 2
c = x.index(1)
# index() will return the first index of 1 in x, so c will be 0
x.remove(1)
# remove the leftmost element that have a value of 1
# x will become [2, 3, 1, 4]
x.sort()
# sort the values of x
# x will become [1, 2, 3, 4]
x.reverse()
# reverse the order of x
# x will become [4, 3, 2, 1]
x.clear()
# x will become []
```

Dictionary Methods Examples

```python
ddict = dict((['a', 1], ['b', 2], ['c', 3]))

ddict.update({'d': 4})
x = ddict.get('a')
# x will be equal to 1, get() is similar to []

keys = ddict.keys()
for key in keys:
    print(key, end = ' ')
print()
# This will print: a b c d

values = ddict.values()
for value in values:
    print(value, end = ' ')
print()
# This will print: 1 2 3 4

ddict.pop('d')

items = ddict.items()
for key, value in items:
    print(key, value)
'''
This will print:
a 1
b 2
c 3
'''
```

Reading Files Examples

```python
with open('foo.txt', 'r') as f:
    first_line = f.readline()
    for a_line in f:
        print(a_line.rstrip('\n'))
```

```
Hello World
Let's learn Python!     (Inside foo.txt)
Python is fun!!!
```

```
Let's learn Python!     Program output
Python is fun!!!
```

Note:
Reading files is like a pointer. After a part of the file is being read, it
won't be repeated again.

Writing to Files Examples

```python
with open('foo.txt', 'w') as f:
    f.write('Hello World\n')
    f.write('Python is fun!')
```

```
Hello World     (Inside foo.txt)
Python is fun!
```

ASCII TABLE

Decimal	Hex	Char		Decimal	Hex	Char		Decimal	Hex	Char		Decimal	Hex	Char
0	0	[NULL]		32	20	[SPACE]		64	40	@		96	60	`
1	1	[START OF HEADING]		33	21	!		65	41	A		97	61	a
2	2	[START OF TEXT]		34	22	"		66	42	B		98	62	b
3	3	[END OF TEXT]		35	23	#		67	43	C		99	63	c
4	4	[END OF TRANSMISSION]		36	24	$		68	44	D		100	64	d
5	5	[ENQUIRY]		37	25	%		69	45	E		101	65	e
6	6	[ACKNOWLEDGE]		38	26	&		70	46	F		102	66	f
7	7	[BELL]		39	27	'		71	47	G		103	67	g
8	8	[BACKSPACE]		40	28	(		72	48	H		104	68	h
9	9	[HORIZONTAL TAB]		41	29	)		73	49	I		105	69	i
10	A	[LINE FEED]		42	2A	*		74	4A	J		106	6A	j
11	B	[VERTICAL TAB]		43	2B	+		75	4B	K		107	6B	k
12	C	[FORM FEED]		44	2C	,		76	4C	L		108	6C	l
13	D	[CARRIAGE RETURN]		45	2D	-		77	4D	M		109	6D	m
14	E	[SHIFT OUT]		46	2E	.		78	4E	N		110	6E	n
15	F	[SHIFT IN]		47	2F	/		79	4F	O		111	6F	o
16	10	[DATA LINK ESCAPE]		48	30	0		80	50	P		112	70	p
17	11	[DEVICE CONTROL 1]		49	31	1		81	51	Q		113	71	q
18	12	[DEVICE CONTROL 2]		50	32	2		82	52	R		114	72	r
19	13	[DEVICE CONTROL 3]		51	33	3		83	53	S		115	73	s
20	14	[DEVICE CONTROL 4]		52	34	4		84	54	T		116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]		53	35	5		85	55	U		117	75	u
22	16	[SYNCHRONOUS IDLE]		54	36	6		86	56	V		118	76	v
23	17	[END OF TRANS. BLOCK]		55	37	7		87	57	W		119	77	w
24	18	[CANCEL]		56	38	8		88	58	X		120	78	x
25	19	[END OF MEDIUM]		57	39	9		89	59	Y		121	79	y
26	1A	[SUBSTITUTE]		58	3A	:		90	5A	Z		122	7A	z
27	1B	[ESCAPE]		59	3B	;		91	5B	[		123	7B	{
28	1C	[FILE SEPARATOR]		60	3C	<		92	5C	\		124	7C		
29	1D	[GROUP SEPARATOR]		61	3D	=		93	5D	]		125	7D	}
30	1E	[RECORD SEPARATOR]		62	3E	>		94	5E	^		126	7E	~
31	1F	[UNIT SEPARATOR]		63	3F	?		95	5F	_		127	7F	[DEL]

Errors

ArithmeticError	Raised when an error occurs in numeric calculations
AssertionError	Raised when an assert statement fails
AttributeError	Raised when attribute reference or assignment fails
Exception	Base class for all exceptions
EOFError	Raised when the input() method hits an "end of file" condition (EOF)
FloatingPointError	Raised when a floating point calculation fails
GeneratorExit	Raised when a generator is closed (with the close() method)
ImportError	Raised when an imported module does not exist
IndentationError	Raised when indentation is not correct
IndexError	Raised when an index of a sequence does not exist
KeyError	Raised when a key does not exist in a dictionary
KeyboardInterrupt	Raised when the user presses Ctrl+c, Ctrl+z or Delete
LookupError	Raised when errors raised cant be found
MemoryError	Raised when a program runs out of memory
NameError	Raised when a variable does not exist
NotImplementedError	Raised when an abstract method requires an inherited class to override the method
OSError	Raised when a system related operation causes an error
OverflowError	Raised when the result of a numeric calculation is too large
ReferenceError	Raised when a weak reference object does not exist
RuntimeError	Raised when an error occurs that do not belong to any specific exceptions
StopIteration	Raised when the next() method of an iterator has no further values
SyntaxError	Raised when a syntax error occurs
TabError	Raised when indentation consists of tabs or spaces
SystemError	Raised when a system error occurs
SystemExit	Raised when the sys.exit() function is called
TypeError	Raised when two different types are combined
UnboundLocalError	Raised when a local variable is referenced before assignment
UnicodeError	Raised when a unicode problem occurs
UnicodeEncodeError	Raised when a unicode encoding problem occurs
UnicodeDecodeError	Raised when a unicode decoding problem occurs
UnicodeTranslateError	Raised when a unicode translation problem occurs
ValueError	Raised when there is a wrong value in a specified data type
ZeroDivisionError	Raised when the second operator in a division is zero

Written by Jansen Ken Pegrasio