3.5.4. Useful Functions

Fig. 3.5.4.1 Photo by Alejandro Piñero Amerio on Unsplash

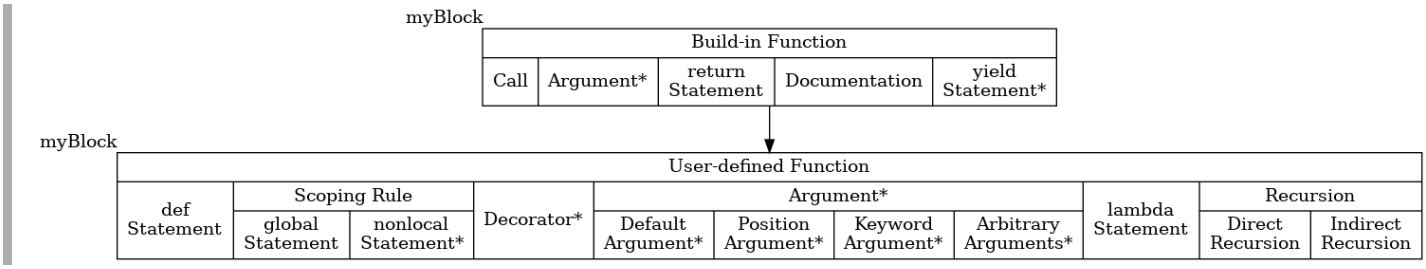
Note

Outline

Note

Roadmap

1. This topic: Function



2. Course: Python 1
3. Subject: Programming
4. Field
    a. Software Engineering (SE)
    b. Computer Science and Information Engineering (CSIE)
    c. Electrical/Electronics Engineering (EE)

3.5.4.1. Overview

1. The following table lists several commonly useful built-in functions in Python.

Function	Description
map()	Apply a function to each item in an iterable (list, tuple, etc)
join()	Used to concatenate sequence elements into a string using a separator.
filter()	Filter items in an iterable based on a predicate function
zip()	Combine two iterables by pairing up elements
byte()	Converts an integer in the range 0 <= x < 256 to a byte object.
enumerate()	Add counter to an iterable and return it
any()	Check if any element of an iterable is True
all()	Check if all elements of an iterable are True
sorted()	Return a sorted list from a given iterable
sum()	Sum the items of an iterable

Function	Description
min()	Return the minimum item of an iterable
max()	Return the maximum item of an iterable
abs()	Return absolute value of a number
round()	Round a floating point number
len()	Get the length of an iterable
str()	Convert object to a string
int()	Convert object to integer
list()	Convert iterable to list
dict()	Create a dictionary
set()	Create a set
slice()	Create a slice object for slicing sequences
reversed()	Reverse an iterable
open()	Open a file and return a file object

2. These built-in functions provide very useful abstractions for common data operations and are widely used in Python programming.

3.5.4.2. map()

1. The map() function in Python applies a function to each item in an iterable (list, tuple, etc) and returns a new iterator with the transformed items.

2. Syntax

```
map(function, iterable)
```

3. The function is applied to each item in the iterable and a map object is returned.

4. Some key points on map():

    a. The function must take a single argument and return a value

    b. The iterable can be any sequence like a list, tuple, etc

    c. The returned map object can be converted to a data structure

    d. map() is lazy. That is, it evaluates items as needed

e. For larger iterables, map() is faster than a loop

3.5.4.2.1. Ex1: Squared

1. Code

Listing 3.5.4.2.1.1 /src/Function/map/Ex1.py

```
'''
author: cph
since: 20230727
'''

if __name__ == '__main__':
    liN = [1, 2, 3, 4, 5]
    loSquar = map(lambda x: x**2, liN)
    print(loSquar)
    print(list(loSquar))
```

2. Output

```
<map object at 0x0000012F653398A0>
[1, 4, 9, 16, 25]
```

3.5.4.2.2. Ex2: Names

1. Code

Listing 3.5.4.2.2.1 /src/Function/map/Ex2.py

```
'''
author: cph
since: 20230727
'''

if __name__ == '__main__':
    lsNames = ['John', 'Sarah', 'Mike']
    loUpper = map(str.upper, lsNames)
    print(loUpper)
    print(list(loUpper))
```

2. Output

```
<map object at 0x0000024E3E7F9900>
['JOHN', 'SARAH', 'MIKE']
```

3.5.4.3. join()

1. The join() method in Python is used to concatenate elements of an iterable (list, tuple, etc) into a string using a separator.

2. Some key points on join():

    a. The iterable can be any sequence: string, list, tuple, etc.

b. The separator string is joined between elements of the given iterable.

c. By default, an empty string separator is used if not specified.

d. join() is called on the separator string, not the iterable.

3.5.4.3.1. Ex1a: List

1. Code

Listing 3.5.4.3.1.1 /src/Function/join/Ex1a.py

```
'''
author: CPH
since: 20230727
'''

if __name__ == '__main__':
    lsStr = ['I', 'Love', 'Python']
    print(''.join(lsStr))
    print(' '.join(lsStr))
    print('.'.join(lsStr))
    print('-'.join(lsStr))
    print('__'.join(lsStr))
```

2. Output

```
ILovePython
I Love Python
I.Love.Python
I-Love-Python
I__Love__Python
```

3.5.4.3.2. Ex1b: List: Char

1. Code

Listing 3.5.4.3.2.1 /src/Function/join/Ex1b.py

```
'''
author: CPH
since: 20230727
'''

if __name__ == '__main__':
    lsStr = ['p', 'y', 't', 'h', 'o', 'n']
    print(''.join(lsStr))
    print(' '.join(lsStr))
    print('.'.join(lsStr))
    print('-'.join(lsStr))
    print('__'.join(lsStr))
```

2. Output

```
python
p y t h o n
p.y.t.h.o.n
```

```
p-y-t-h-o-n
p__y__t__h__o__n
```

3.5.4.4. filter()

1. The filter() function in Python applies a test function to each item in an iterable, and returns a new iterable containing only the items for which the function returned True.
2. Syntax

```
filter(function, iterable)

# function: The test function which returns True or False
# iterable: The sequence to filter
```

3. Some key points on filter():

    a. The function can be a lambda or regular function.

    b. Any iterable type like lists, tuples, sets can be filtered.

    c. Elements for which the function returns True are retained.

    d. The original iterable is unchanged.

4. filter() returns a lazy iterator - it evaluates elements on demand.

3.5.4.4.1. Ex1: List

1. Code

Listing 3.5.4.4.1.1 /src/Function/filter/Ex1.py

```
'''
author: CPH
since: 20230727
'''

if __name__ == '__main__':
    liNum = [1, 2, 3, 4, 5, 6]
    loRet = filter(lambda x: x % 2 == 0, liNum)
    print(loRet)
    print(list(loRet))

```

2. Output

```
<filter object at 0x0000023F895C98D0>
[2, 4, 6]
```

3.5.4.4.2. Ex2: Names

1. Code

Listing 3.5.4.4.2.1 /src/Function/filter/Ex2.py

```
'''
author: CPH
since: 20230727
'''

if __name__ == '__main__':
    lsNames = ['Andy', 'Rong', 'Jessie', 'Coco', 'Sean']
    loLong = filter(lambda x: len(x) > 4, lsNames)
    print(loLong)
    print(list(loLong))

```

2. Output

```
<filter object at 0x000002197A3098A0>
['Jessie']
```

3.5.4.5. byte()

1. The byte() method in Python converts an integer in the range 0 <= x < 256 to a byte object.
2. A byte object is an immutable sequence of integers in the range 0 <= x < 256.
3. To create a bytes object from a literal, use a [b] prefix.

```
b = b'Hello, Python...'
```

4. Some notes on bytes vs strings:

    a. Strings in Python 3 are unicode by default.

    b. bytes represent raw 8-bit values.

    c. Useful for binary data, network streams, files etc.

    d. Immutable like strings.

    e. Supports similar operations like indexing, slicing etc.

5. Summary

    a. byte() can convert integers 0-255 to bytes, and creates bytes objects which store raw byte values.

    b. Useful for binary data manipulation.

3.5.4.5.1. Ex1

1. Code

Listing 3.5.4.5.1.1 /src/Function/byte/Ex1.py

```
'''
author: CPH
since: 20230731
'''

if __name__ == '__main__':
```

```
    b = bytes(4)                # Convert int to bytes
    print(f'b: {b}')

    sStr = 'Hello, Python...'   # Convert string to bytes
    b = bytes(sStr, encoding='utf-8')
    print(f'b: {b}')

    print('Byte Iteration: ', end='')
    for c in b:                 # Iterate through bytes
      print(c, end=' ')

    print(f'\nb[0]: {b[0]}')    # Index bytes object
    print(f'len(b): {len(b)}')  # Length of bytes
```

2. Output

```
b: b'\x00\x00\x00\x00'
b: b'Hello, Python...'
Byte Iteration: 72 101 108 108 111 44 32 80 121 116 104 111 110 46 46 46
b[0]: 72
len(b): 16
```

3.5.4.6. enumerate()

1. The enumerate() function in Python allows you to loop over an iterable and get the index position and value during iteration.
2. Syntax

```
for i, v in enumerate(list):
    print(i, v)
```

3. enumerate() takes two parameters:
    a. iterable: sequence to create enumerate object from.
    b. start (optional): starting index value, default is 0.
4. Some key points about enumerate()
    a. Returns an enumerate object that produces a tuple with index and value
    b. Useful when you need to access the index while iterating
    c. Provides concise syntax for iterating with indexes
    d. start param allows setting a different starting index
    e. Indexes start at 0 by default
5. Summary
    a. enumerate() is very useful where you need both the index and value of items while iterating.
    b. A cleaner alternative to manually tracking indexes.

3.5.4.6.1. Ex1: Names

1. Code

Listing 3.5.4.6.1.1 /src/Function/enumerate/Ex1.py

```
'''
author: CPH
since: 20230731
'''

if __name__ == '__main__':
    lsNames = ['Andy', 'Rong', 'Jessie', 'Coco', 'Sean']

    for i, name in enumerate(lsNames):
      print(i, name)
```

2. Output

```
0 Andy
1 Rong
2 Jessie
3 Coco
4 Sean
```

3.5.4.7. slice()

1. The slice() function in Python returns a slice object that can be used to slice sequences like lists, tuples, strings etc.
2. Syntax

```
slice(start, stop, step)
```

3. This creates a slice object that can slice from start up to but not including stop, stepping by the given step.
4. The start, stop and step parameters are optional:
    a. start: Starting index, defaults to 0.
    b. stop: Ending index, defaults to the length of the sequence.
    c. step: Step size, defaults to 1.
5. Using slice() is more flexible than just specifying indexes directly.
    a. Omitting start/stop defaults to the sequence start/end.
    b. Negative indexes can be used for reverse slicing.
    c. step parameter can stride through values.
6. Some common uses of slice().
    a. Extracting subsequences e.g. a[slice(5, 10)].
    b. Reversing sequences e.g. a[slice(None, None, -1)].
    c. Cloning sequences a[slice(None)].
    d. Passing slice objects to functions like batching.

7. Summary

    a. slice() creates slice objects that provide a flexible way to slice Python sequences.

3.5.4.7.1. Ex1: String vs. List

1. Code

Listing 3.5.4.7.1.1 /src/Function/slice/Ex1.py

```
'''
author: cph
since:  20230804
'''

if __name__ == '__main__':
    s = 'Hello, Python...'
    print(s[slice(1, 4)])

    n = [1, 2, 3, 4, 5]
    print(n[slice(1, 4, 2)])
    print(n[slice(None, None, -1)])
```

2. Output

```
ell
[2, 4]
[5, 4, 3, 2, 1]
```

3.5.4.7.2. Ex2: Negative Step

1. Code

Listing 3.5.4.7.2.1 /src/Function/slice/Ex2.py

```
'''
author: CPH
since:  20230804
'''

if __name__ == '__main__':
    s = 'Hello, Python...'
    print(f'  -+++++++++++++++ << slice() Index')
    print(f'i:10123456789012345 << String + Index')
    print(f's: {s}')
    print(f'-: 6543210987654321 << String - Index')
    print()
    print('--[C1: slice(): + Step; + Idx ]-----------')
    print(slice(0, 5, 1))
    print(slice(5, 0, 1))
    print(slice(None, None, 1))
    print()
    print('--[C2: slice(): + Step; - Idx ]-----------')
    print(slice(0, -5, 1))
    print(slice(-5, 0, 1))
    print()
    print('--[C3: slice(): - Step; + Idx ]-----------')
    print(slice(0, 5, -1))
```

```
    print(slice(5, 0, -1))
    print(slice(None, None, -1))
    print()
    print('--[C4: slice(): - Step; - Idx ]-----------')
    print(slice(0, -5, -1))
    print(slice(-5, 0, -1))
    print()
    print('--[D1: slice().indices(): + Step; + Idx ]---------')
    print(slice(0, 5, 1).indices(len(s)))
    print(slice(5, 0, 1).indices(len(s)))
    print(slice(None, None, 1).indices(len(s)))
    print()
    print('--[D2: slice().indices(): + Step; - Idx ]---------')
    print(slice(0, -5, 1).indices(len(s)))
    print(slice(-5, 0, 1).indices(len(s)))
    print()
    print('--[D3: slice().indices(): - Step; + Idx ]---------')
    print(slice(0, 5, -1).indices(len(s)))
    print(slice(5, 0, -1).indices(len(s)))
    print(slice(None, None, -1).indices(len(s)))
    print()
    print('--[D4: slice().indices(): - Step; - Idx ]---------')
    print(slice(0, -5, -1).indices(len(s)))
    print(slice(-5, 0, -1).indices(len(s)))
    print()
    print('--[E1: String.slice(): + Step; + Idx ]-----------')
    print(s[slice(0, 5, 1)])
    print(s[slice(5, 0, 1)])
    print(s[slice(None, None, 1)])
    print()
    print('--[E2: String.slice(): + Step; - Idx ]-----------')
    print(s[slice(0, -5, 1)])
    print(s[slice(-5, 0, 1)])
    print()
    print('--[E3: String.slice(): - Step; + Idx ]-----------')
    print(s[slice(0, 5, -1)])
    print(s[slice(5, 0, -1)])
    print(s[slice(None, None, -1)])
    print()
    print('--[E4: String.slice(): - Step; - Idx ]-----------')
    print(s[slice(0, -5, -1)])
    print(s[slice(-5, 0, -1)])
```

2. Output

```
  -+++++++++++++++ << slice() Index
i:10123456789012345 << String + Index
s: Hello, Python...
-: 6543210987654321 << String - Index

--[C1: slice(): + Step; + Idx ]-----------
slice(0, 5, 1)
slice(5, 0, 1)
slice(None, None, 1)

--[C2: slice(): + Step; - Idx ]-----------
slice(0, -5, 1)
slice(-5, 0, 1)

--[C3: slice(): - Step; + Idx ]-----------
slice(0, 5, -1)
slice(5, 0, -1)
slice(None, None, -1)

--[C4: slice(): - Step; - Idx ]-----------
slice(0, -5, -1)
```

```
slice(-5, 0, -1)

--[D1: slice().indices(): + Step; + Idx ]---------
(0, 5, 1)
(5, 0, 1)
(0, 16, 1)

--[D2: slice().indices(): + Step; - Idx ]---------
(0, 11, 1)
(11, 0, 1)

--[D3: slice().indices(): - Step; + Idx ]---------
(0, 5, -1)
(5, 0, -1)
(15, -1, -1)

--[D4: slice().indices(): - Step; - Idx ]---------
(0, 11, -1)
(11, 0, -1)

--[E1: String.slice(): + Step; + Idx ]-----------
Hello

Hello, Python...

--[E2: String.slice(): + Step; - Idx ]-----------
Hello, Pyth


--[E3: String.slice(): - Step; + Idx ]-----------

,olle
...nohtyP ,olleH

--[E4: String.slice(): - Step; - Idx ]-----------

ohtyP ,olle
```

Important

String Example

Listing 3.5.4.7.2.2 /src/Function/slice/Ex2a.py

```
'''
author: CPH
since:  20230804
'''

if __name__ == '__main__':
    s = 'Hello, Python...'
    print(f'+: 0123456789012345 << String + Index')
    print(f's: {s}')
    print(f'-: 6543210987654321 << String - Index')
    print()
    print('--[A1: + Step; + Idx ]------------')
    print(s[0: 5: 1])
    print(s[5: 0: 1])
    print(s[ : 5: 1])
    print(s[5:  : 1])
    print(s[ :  : 1])
    print()
    print('--[A2: + Step; - Idx ]------------')
    print(s[ 0: -5: 1])
    print(s[-5:  0: 1])
    print(s[  : -5: 1])
    print(s[-5:   : 1])
    print()
    print('--[B1: - Step; + Idx ]------------')
    print(s[0: 5: -1])
    print(s[5: 0: -1])
    print(s[ : 5: -1])
    print(s[5:  : -1])
    print(s[ :  : -1])
    print()
    print('--[B2: - Step; - Idx ]------------')
    print(s[ 0: -5: -1])
    print(s[-5:  0: -1])
    print(s[  : -5: -1])
    print(s[-5:   : -1])
```

```
+: 0123456789012345 << String + Index
s: Hello, Python...
-: 6543210987654321 << String - Index

--[A1: + Step; + Idx ]------------
Hello

Hello
, Python...
Hello, Python...

--[A2: + Step; - Idx ]------------
Hello, Pyth

Hello, Pyth
on...

--[B1: - Step; + Idx ]------------

,olle
...nohtyP
,olleH
```

```
...nohtyP ,olleH

--[B2: - Step; - Idx ]------------

ohtyP ,olle
...n
ohtyP ,olleH
```

Note

1. Start: 20170719
2. System Environment

Listing 3.5.4.7.2.3 requirements.txt

```
sphinx==7.1.2                           # Sphinx
graphviz>=0.20.1                        # Graphviz
sphinxbootstrap4theme>=0.6.0            # Theme: Bootstrap
sphinx-material>=0.0.35                 # Theme: Material
sphinxcontrib-plantuml>=0.25            # PlantUML
sphinxcontrib.bibtex>=2.5.0             # Bibliography
sphinx-autorun>=1.1.1                   # ExecCode: pycon
sphinx-execute-code-python3>=0.3        # ExecCode
btd.sphinx.inheritance-diagram>=2.3.1   # Diagram
sphinx-copybutton>=0.5.1                # Copy button
sphinx_code_tabs>=0.5.3                 # Tabs
sphinx-immaterial>=0.11.3               # Tabs

#-----------------------------------------------------
#-- Library Upgrade Error by Library Itself
#   >> It needs to fix by library owner
#   >> After fixed, we need to try it later
#-----------------------------------------------------
pydantic==1.10.10                       # 2.0: sphinx compiler error, 20230701

#-----------------------------------------------------
#-- Minor Extension
#-----------------------------------------------------
sphinxcontrib.httpdomain>=1.8.1         # HTTP API

#sphinxcontrib-blockdiag>=3.0.0         # Diagram: block
#sphinxcontrib-actdiag>=3.0.0           # Diagram: activity
#sphinxcontrib-nwdiag>=2.0.0            # Diagram: network
#sphinxcontrib-seqdiag>=3.0.0           # Diagram: sequence

#-----------------------------------------------------
#-- Still Wait For Upgrading Version
#-----------------------------------------------------

#-----------------------------------------------------
#-- Still Under Testing
#-----------------------------------------------------
#numpy>=1.24.2                          # Figure: numpy

#-----------------------------------------------------
#-- NOT Workable
#-----------------------------------------------------
#sphinxcontrib.jsdemo==0.1.4    # ExecCode: Need replace add_js_file()
#jupyter-sphinx==0.4.0          # ExecCode: Need gcc compiler
#sphinxcontrib.slide==1.0.0     # Slide: Slideshare
#hieroglyph==2.1.0              # Slide: make slides
#matplotlib>=3.7.1              # Plot: Need Python >= v3.8
#manim==0.17.2                  # Diagram: scipy, numpy need gcc
#sphinx_diagrams==0.4.0         # Diagram: Need GKE access
#sphinx-tabs>=3.4.1                     # Tabs: Conflict w/ sphinx-material
```