reference

Hack Chyson

August 16, 2019

Contents

1 import **3**

2 list **3**

3 string **3**

4 tuple **3**

5 mapping unpacking **3**

6 sequence unpacking **3**

7 private method **4**

8 hashable **4**

9 list comprehension **4**

10 set **4**

11 data create with data type **5**

12 program format **5**

13 path separator **6**

14 random choice **6**

15 random sample **6**

16 help **6**

17 type **7**

18 sort **7**

19 plateform **7**

20 file number **7**

21 list find **7**

22 if **8**

23 loop **9**

24 try . . . catch **9**

25 raise exception **9**

26 function **10**

27 lambda **10**

28 assert **10**

29 writing text to files **10**

30 capture output intended to go to a file **10**

31 command line options **11**

32 doctest **11**

33 unit test **11**

34 eval **11**

35 special attributes **12**

36 @staticmethod and @classmethod **13**

1 import

```
import ...
import ... as ...
from ... import ...
from ... import *
```

2 list

[]

3 string

"a"
'a'

4 tuple

'a','b'
('a','b')

5 mapping unpacking

the mapping unpacking operator is ** and it can be applied to a mapping to produce a key-value list.
**locals()

6 sequence unpacking

Any iterable can unpacked using the sequence unpacking operator (*)

first, *rest = [1,2,3,4,5]

def prouct(a, b, c):
return a * b * c

l = [1,2,3]
product(*l)

7 private method

the method name begins with a leading understore.

Sale = collections.namedtuple('Sale', 'productid price')
sale = Sale('book', 100)
'{book} {price}'.format(**sale._asdict())

8 hashable

Hashable objects are objects which have a __hash__() speical method whose return value is always the same throughout the object's lifetime, and which can be compared for equality using the __eq__() special method.

All the built-in immutable data types are hashable.
The built-in mutable data types, such as dict, list, and set are not hashable.

9 list comprehension

[expression for item in iterable if condition]

10 set

{}

11 data create with data type

- with one argument, create a empty object.
- one same data type argument, a shallow copy
- one different data type argument, a conversion attempt
- two or more, depends

for example:
list()
list('hello')
list([1])

str()
str(1)
str('hello')

12 program format

1. shebang
2. docstring
   (a) brief description
   (b) one black line
   (c) description
   (d) example

[1]DEFINITION NOT FOUND.

3. import

   (a) built-in

   (b) standard library

   (c) custom

4. all

5. code

13 path separator

```
import os
path.replace("/",os.sep)
```

14 random choice

```
l = [1,2,3,4]
import random
random.choice(l)
```

15 random sample

```
import random
l = list(range(100))
print(random.sample(l,30))  # unique element
```

16 help

```
help(iter)
```

17 type

```
a = 1
type(a)
```

18 sort

```
x = []
for i in zip(range(-10, 0, 1), range(0, 10, 2), range(1, 10, 2)):
    x += i
print(x)  # [-10, 0, 1, -9, 2, 3, -8, 4, 5, -7, 6, 7, -6, 8, 9]

y = []
for i in zip(range(-10, 0, 1), range(0, 10, 2), range(1, 10, 2)):
    y.append(i)
print(y)  # [(-10, 0, 1), (-9, 2, 3), (-8, 4, 5), (-7, 6, 7), (-6, 8, 9)]

print(sorted(x))  # [-10, -9, -8, -7, -6, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(sorted(x, reverse=True))  # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -6, -7, -8, -9, -10]
print(sorted(x, key=abs))  # [0, 1, 2, 3, 4, 5, 6, -6, -7, 7, -8, 8, -9, 9, -10]
```

19 plateform

```
offset = 20 if sys.platform.startswith('win') else 10
```

20 file number

```
print('{} file{}'.format((count if count != 0 else 'no'), ('s' if count != 1 else '')))
```

21 list find

like list find method.

while version:

```
def list_find(lst, target):
    index = 0
    while index < len(lst):
        if lst[index] == target:
            break
        index += 1
    else:
        index = -1
    return index
```

loop version:

```
def list_find(lst,target):
    for index, x in enumerate(lst):
        if x == target:
            break
    else:
        index = -1
    return index
```

exception version:

```
def list_find(lst,target):
    try:
        index = lst.index(target)
    except ValueError:
        index = -1
    return index
```

22 if

conditional branch statement:

```
if boolean_expression1:
    suite1
elif boolean_expression2:
    suite2
...
elif boolean_expressionN:
    suiteN
else:
    else_suite
```

There can be zero or more elif clauses, and the final else clause is optional.

conditional expression:

```
expression1 if boolean_expression else expression2
```

23 loop

```
while boolean_expression:
    while_suite
else:
    else_suite

for expression in iterable:
    for_suite
else:
    else_suite
```

24 try ... catch

```
try:
    try_suite
except exception_group1 as variable1:
    except_suite1
...
except exception_groupN as variableN:
    except_suiteN
else:
    else_suite
finally:
    finally_suite
```

25 raise exception

```
raise exception(args)
raise exception(args) from original_exception
raise
```

26 function

```
def functionName(parameters):
    suite
```

27 lambda

```
lambda parameters: expression
```

28 assert

```
assert boolean_expression, optional_expression
```

29 writing text to files

Python provides tow different ways of writing text to files.

1. use a file object's write() method

2. use the print() function

```
import sys
sys.stdout.write("message\n")
print("message", file=sys.stdout)

# to restore back to stdout
sys.stdout = sys.__stdout__
```

30 capture output intended to go to a file

```
import io
import sys

sys.stdout = io.StringIO()
print('hello')
print('world')
content = sys.stdout.getvalue()
```

```
sys.stdout = sys.__stdout__
print(content)
```

31 command line options

```
parser = optparse.OptionParser()
parser.set_usage("%prog inputfile outputfile [options]")
parser.add_option("-m", "--mode", dest="mode",
                  help="available values: encrypt|enc|decrypt|dec [default: %default]"
parser.add_option('-k', '--key', dest='key',
                  help='the key for encryption and decryption [default: %default')
parser.set_defaults(mode="enc", key='123456')
opts, args = parser.parse_args()

inputfile = args[0]
outputfile = args[1]
mode = opts.mode
key = opts.key
```

"%default" rext replaced with the option's default value;
the options are available using the "dest" names.

If an error occurs when parsing the command line, the optparse parser will call sys.exit(2).

32 doctest

```
if __name__ == "__main__":
    import doctest
    doctest.testmod() # test module
```

33 unit test

```
import unittest
```

34 eval

```
import Shape
```

```
p = Shape.Point(3, 9)
print(repr(p))  # Point(3, 9)

# We must give the module name when eval()ing if we used import Shape.
# if from Shape import Point is used, it is used necessary
q = eval(p.__module__ + "." + repr(p))
print(repr(q))  # Point(3, 9)
```

35 special attributes

$\underline{\underline{\text{name}}}$
$\underline{\underline{\text{module}}}$
$\underline{\underline{\text{class}}}$

@property
@staticmethod
@classmethod

__lt__(self, other)	<
__le__(self, other)	<=
__eq__(self, other)	==
__ne__(self, other)	!=
__ge__(self, other)	>=
__gt__(self, other)	>

__bool__(self)
__format__(self,format_spec)
__hash__(self)
__init__(self,args)
__new__(cls,args)
__repr__(self)
__str__(self)

__abs__(self)
__pos__(self)
__add__(self)
__iadd__(self)
__radd__(self)
__xor__(self)
__ixor__(self)
__rxor__(self)

36 @staticmethod and @classmethod

@classmethod must have a reference to a class object as the first parameter, whereas @staticmethod can have no parameters at all.

```
# The first parameter is the object instance reference.
def instancemethod_(self, ...)
    suit


# The first parameter is the class reference.
@classmethod
def classmethod_(cls, ...)
    suit


# There is no reference to the instance or class.
@staticmethod(...)
    suit
```