LECTURE 19: THE GOODIES

Hung-Yu Wei

Some more techniques

(1) Conditional expressions

(2) List comprehensions

(3) Generator expressions

(4) any & all

(5) Set

(6) Counter

(7) Defaultdict

(8) Named tuple

(9) Gathering keyword args

(1) Conditional expressions

- If …else…

```
if x > 0:
    y = math.log(x)
else:
    y = float('nan')
```

NaN: Not a Number

(1) Conditional expressions

- *Write "if… else " in single line*

```
y = math.log(x) if x > 0 else float('nan')
```

Examples: conditional expression

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

```
def factorial(n):
    return 1 if n == 0 else n * factorial(n-1)
```

```
def __init__(self, name, contents=None):
    self.name = name
    if contents == None:
        contents = []
    self.pouch_contents = contents
```

```
def __init__(self, name, contents=None):
    self.name = name
    self.pouch_contents = [] if contents == None else contents
```

(2) List comprehensions

```
def capitalize_all(t):
    res = []
    for s in t:
        res.append(s.capitalize())
    return res
```

```
def capitalize_all(t):
    return [s.capitalize() for s in t]
```

More Example: List comprehensions

```
def only_upper(t):
    res = []
    for s in t:
        if s.isupper():
            res.append(s)
    return res
```

```
def only_upper(t):
    return [s for s in t if s.isupper()]
```

(3) Generator expressions

```
g = (x**2 for x in range(10))
print(g)
print(next(g))
print(next(g))
print(next(g))
print(next(g))
```

```
<generator object <genexpr> at 0x0000000005057FC0>
0
1
4
9
```

```
for val in g:
    print(val)
```

```
16
25
36
49
64
81
```

```
next(g)
```

```
---------------------------------------------
StopIteration                             Trac
<ipython-input-16-e734f8aca5ac> in <module>()
----> 1 next(g)

StopIteration:
```

More example: generator expressions

```
1  sum(x**2 for x in range(5))
```

```
30
```

0+1+4+9+16=30

(4) any & all

- Any
  - *takes a sequence of boolean values and returns True if any of the values are True.*
  - *often used with generator expressions*

```
1 any([False, False, True])
```

```
True
```

```
1 any(letter == 't' for letter in 'monty')
```

```
True
```

```
1 any(letter == 'x' for letter in 'monty')
```

```
False
```

- All
  - *returns True if every element of the sequence is True*

```
1 all([False, False, True])
```

```
False
```

```
1 all(letter == 't' for letter in 'ttt')
```

```
True
```

(5) Set

```
def subtract(d1, d2):
    res = dict()
    for key in d1:
        if key not in d2:
            res[key] = None
    return res
```

```
def subtract(d1, d2):
    return set(d1) - set(d2)
```

Set difference

Example: Set

```
def has_duplicates(t):
    d = {}
    for x in t:
        if x in d:
            return True
        d[x] = True
    return False
```

```
def has_duplicates(t):
    return len(set(t)) < len(t)
```

len()

Set: subset <=

```
def uses_only(word, available):
    for letter in word:
        if letter not in available:
            return False
    return True
```

```
def uses_only(word, available):
    return set(word) <= set(available)
```

Check whether it is subset

(6) Counter

- **Counter**
  - *Like a set but an element appears more than once*
  - *Multiset*
    - https://en.wikipedia.org/wiki/Multiset
- Module: **collections**

```
>>> from collections import Counter
>>> count = Counter('parrot')
>>> count
Counter({'r': 2, 't': 1, 'o': 1, 'p': 1, 'a': 1})
```

```
>>> count['d']
0
```

Example: Counter

```
def is_anagram(word1, word2):
    return Counter(word1) == Counter(word2)
```

- most_common()
  - *Most common list of value-frequency pairs*

```
>>> count = Counter('parrot')
>>> for val, freq in count.most_common(3):
...     print(val, freq)
r 2
p 1
a 1
```

(7) Defaultdict

- **defaultdict**
  - ***collections*** *module*
  - *like a dictionary*
  - *if you access a key that doesn't exist, it can generate a new value*

```
1  from collections import defaultdict
2  d = defaultdict(list)
3  print(d)
4  t = d['Key1']
5  print(t)
6  t.append('value2')
7  print(d)
```

```
defaultdict(<class 'list'>, {})
[]
defaultdict(<class 'list'>, {'Key1': ['value2']})
```

(8) Named tuple

- **Name tuple** for simpler description of a class
  - *(1) name of the class*
  - *(2) list of the attributes*

```
from collections import namedtuple
Point = namedtuple('Point', ['x', 'y'])
```

```
class Point:

    def __init__(self, x=0, y=0):
        self.x = x
        self.y = y

    def __str__(self):
        return '(%g, %g)' % (self.x, self.y)
```

對照組

Example: Name Tuple

```
1  from collections import namedtuple
2
3  class Point:
4      def __init__(self, x=0, y=0):
5          self.x = x
6          self.y = y
7      def __str__(self):
8          return '(%g, %g)' % (self.x, self.y)
9
10
11 Point = namedtuple('Point', ['x', 'y'])
12
13 p = Point(1, 2)
14 print(p)
```

```
Point(x=1, y=2)
```

```
>>> p.x, p.y
(1, 2)
```

```
>>> p[0], p[1]
(1, 2)
```

```
>>> x, y = p
>>> x, y
(1, 2)
```

(9) Gathering keyword args

- Review: Variable-length argument tuples
  - ** gathers arguments into a tuple*

```
1  def printall(*args):
2      print(args)
3
4  printall(1, 2.0, '3')
```

```
(1, 2.0, '3')
```

```
1  # error
2  printall(1, 2.0, third='3')
```

Type Error

** operator

- To gather keyword arguments, use the ** operator

```
1  def printall_new(*args, **kwargs):
2      print(args, kwargs)
3
4  printall_new(1, 2.0, third='3')
```

```
(1, 2.0) {'third': '3'}
```

** operator

```
1  class Point:
2
3      def __init__(self, x=0, y=0):
4          self.x = x
5          self.y = y
6
7      def __str__(self):
8          return '(%g, %g)' % (self.x, self.y)
9
10
11 d = dict(x=1, y=2)
12 print(d)
13 p=Point(**d)
14 print(p)
```

```
{'x': 1, 'y': 2}
(1, 2)
```

Reading

- Chapter 19 in textbook “Think Python”