Tutorial 4: Data Structures and Control Flow

BY ZIQI GAO & YU WU

Data Structures

LISTS, TUPLES, DICTIONARIES AND SETS

Problem 1

Create a list of squares.

- [0, 1, 4, 9, …]
- When really short (like 0^2 to 5^2), why not create by hand?
- But what if to 100^2, 1000^2 , … ?

Problem 1

Create a list of squares.

```
>>> squares = []
>>> for x in range(10):
...     squares.append(x**2)
...
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Not elegant!

Problem 1

Create a list of squares.

Try this:

```
1 squares = [x**2 for x in range(10)]
```

It is list comprehension.

List Comprehension

Something more…

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Problem 2

Transpose a matrix in the form of nested list.

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

Problem 2

Transpose a matrix in the form of nested list.

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

- Nested for loop?

Problem 2

Transpose a matrix in the form of nested list.

Try nested list comprehension:

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

Problem 3

Which of the following cannot construct a tuple?

```
>>> t = ()
>>> t = ('')
>>> t = ('hi')
>>> t = ('hi',)
>>> t = ('hi', 'hello')
>>> t = ('hi', 'hello',)
>>> t = 'hi'
>>> t = 'hi',
>>> t = 'hi', 'hello'
>>> t = 'hi', 'hello',
>>> t = (x**2 for x in range(10))
>>> t = tuple()
>>> t = tuple(x**2 for x in range(10))
>>> t = tuple([x**2 for x in range(10)])
```

Problem 3

Which of the following cannot construct a tuple?

```
1  >>> t = ()
2  >>> t = ('') ✖
3  >>> t = ('hi') ✖
4  >>> t = ('hi',)
5  >>> t = ('hi', 'hello')
6  >>> t = ('hi', 'hello',)
7  >>> t = 'hi' ✖
8  >>> t = 'hi',
9  >>> t = 'hi', 'hello' # tuple packing
10 >>> t = 'hi', 'hello',
11 >>> t = (x**2 for x in range(10)) ✖ Try this (line 11) by yourself!
12 >>> t = tuple()
13 >>> t = tuple(x**2 for x in range(10))
14 >>> t = tuple([x**2 for x in range(10)])
```

Tuple Packing & Sequence Unpacking

```
>>> t = 'hi', 'hello' # tuple packing
```

```
>>> s, t = ('hi', 'hello') # sequence unpacking
```

Tuple Packing & Sequence Unpacking

```
>>> t = 'hi', 'hello' # tuple packing
```

```
>>> s, t = ('hi', 'hello') # sequence unpacking
```

How about this?

```
>>> s, t = ['hi', 'hello']
```

Tuple Packing & Sequence Unpacking

```
>>> t = 'hi', 'hello' # tuple packing
```

```
>>> s, t = ('hi', 'hello') # sequence unpacking
```

How about this?

```
>>> s, t = ['hi', 'hello']
```

Even this!

```
>>> s, t = 'hi'
```

Sequences in Python

In 3.7.7:

- List
- Tuple
- Range
- String
- …

Many interesting similarity…

- Index, slice, insert, remove, …
- Iterable
- See https://docs.python.org/3.7/library/stdtypes.html#typesseq for details

Keep updating…

Problem 4

Which of the following cannot construct a dictionary?

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel = dict()
>>> tel = dict(['jack', 'sape'])
>>> tel = dict(['jack', 4098], ['sape': 4139])
>>> tel = dict([['jack', 4098], ['sape', 4139]])
>>> tel = dict([('jack', 4098), ('sape', 4139)])
>>> tel = dict((('jack', 4098), ('sape', 4139)))
>>> tel = {x: x**2 for x in (2, 4, 6)}
>>> tel = dict(sape=4139, guido=4127, jack=4098)
```

Problem 4

Which of the following cannot construct a dictionary?

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel = dict()
>>> tel = dict(['jack', 'sape'])✖
>>> tel = dict(['jack', 4098], ['sape': 4139])✖
>>> tel = dict([['jack', 4098], ['sape', 4139]])
>>> tel = dict([('jack', 4098), ('sape', 4139)])
>>> tel = dict((('jack', 4098), ('sape', 4139)))
>>> tel = {x: x**2 for x in (2, 4, 6)}
>>> tel = dict(sape=4139, guido=4127, jack=4098) # only work when the keys are strings
```

Problem 5

How to loop through dictionaries?

Problem 5

How to loop through dictionaries?

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}

>>> for i in knights:
...     print(i)
...
gallahad
robin
```

Problem 5

How to loop through dictionaries?

```
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave

>>> for k in knights.keys():
...     print(i)
...
gallahad
robin

>>> for v in knights.values():
```

A Tip about Loop

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

A Tip about Loop

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
>>> for i in enumerate(['tic', 'tac', 'toe']):
...     print(i)
...
(0, 'tic')
(1, 'tac')
(2, 'toe')
```

Problem 6

How to avoid duplication of elements in a list?

- Check when add?

Problem 6

How to avoid duplication of elements in a list?

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> basket = {}
>>> basket = set()
```

Problem 6

How to avoid duplication of elements in a list?

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> basket = {}                        # this is an empty dictionary!
>>> basket = set()
```

Problem 7

How to compare the difference of the elements between two sequences?

And the common part? The sum?

Problem 7

How to compare the difference of the elements between two sequences?

And the common part? The sum?

Try Sets!

Problem 7

How to compare the difference of the elements between two sequences?

And the common part? The sum?

```
>>> a = set('abracadabra')                  # What if a = set([1, 1+0j, True])?
>>> b = set('alacazam')
>>> a                                       # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                                   # letters in a but not in b
{'r', 'd', 'b'}
>>> a | b                                   # letters in a or b or both
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                                   # letters in both a and b
{'a', 'c'}
>>> a ^ b                                   # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Problem 8

Which one is bigger on each line?

```
1  (1, 2, 3)                 (1, 2, 4)
2  [1, 2, 3]                 [1, 2, 4]
3  'ABC'   'C'   'Pascal'    'Python'
4  (1, 2, 3, 4)              (1, 2, 4)
5  (1, 2)                    (1, 2, -1)
6  (1, 2, 3)                 (1.0, 2.0, 3.0)
7  (1, 2, ('aa', 'ab'))      (1, 2, ('abc', 'a'), 4)
```

Problem 8

Which one is bigger on each line?

```
1 (1, 2, 3)                < (1, 2, 4)
2 [1, 2, 3]                < [1, 2, 4]
3 'ABC' < 'C' < 'Pascal' < 'Python'
4 (1, 2, 3, 4)             < (1, 2, 4)
5 (1, 2)                   < (1, 2, -1)
6 (1, 2, 3)               == (1.0, 2.0, 3.0)
7 (1, 2, ('aa', 'ab'))     < (1, 2, ('abc', 'a'), 4)
```

Problem 8

Which one is bigger on each line?

What about sets? Which lines are not true?

```
1 {1, 2, 3}          < {1, 2, 4}
2 {1, 2, 3}          > {1, 2, 4}
3 {1, 2, 3}         == {1, 2, 4}
4 {1, 2}             < {1, 2, 4}
5 {1, 2}            <= {1, 2, 4}
6 {1, 2, 4}         <= {1, 2, 4}
```

Problem 8

Which one is bigger on each line?

What about sets? Which lines are not true?

```
1 {1, 2, 3}          < {1, 2, 4}     # differ from sequences
2 {1, 2, 3}          > {1, 2, 4}
3 {1, 2, 3}         == {1, 2, 4}
4 {1, 2}             < {1, 2, 4}
5 {1, 2}            <= {1, 2, 4}
6 {1, 2, 4}         <= {1, 2, 4}
```

Problem 8

Which one is bigger on each line?

What about sets? Which lines are not true?

```
1 {1, 2, 3}          < {1, 2, 4} ✖  # differ from sequences
2 {1, 2, 3}          > {1, 2, 4} ✖
3 {1, 2, 3}         == {1, 2, 4} ✖
4 {1, 2}             < {1, 2, 4}
5 {1, 2}            <= {1, 2, 4}
6 {1, 2, 4}         <= {1, 2, 4}
```

Control Flow

MODIFICATION WHEN LOOPING, ELSE CLAUSES IN LOOP, PASS STATEMENT, LAMBDA AND FUNCTION

0. Warm-up exercise

What is the output?

```
1 mysum = 0
2 for i in range(5, 11, 2):
3     mysum += i
4     if mysum == 5:
5         break
6     mysum += 1
7 print(mysum)
```

A 6 B 5 C 21 D 23

0. Warm-up exercise

What is the output?

```
1 mysum = 0
2 for i in range(5, 11, 2):
3     mysum += i
4     if mysum == 5:
5         break
6     mysum += 1
7 print(mysum)
```

A 6 B 5 C 21 D 23

1. Modification when looping

Code that modifies a collection while iterating over that same collection can be tricky to get right.

```
1 a = [1,2,1,2,1,1,1]
2 for i in a:
3     if i == 1:
4         a.remove(i)
5 print(a) # a = [2,2,1]
```

Why? When 3rd 1 is removed, the 4th 1 is moved to left, loop will miss this 1 and go to the 5th 1.

1. Modification when looping

Code that modifies a collection while iterating over that same collection can be tricky to get right.

```
>>> a = [0,1,2,3,4,5]
>>> for i in a:
...     a.remove(i)
...     print(a)
...
[1, 2, 3, 4, 5]
[1, 3, 4, 5]
[1, 3, 5]
```

1. Modification when looping

Instead, it is usually more straight-forward to **loop over a copy of the collection** or to create a new collection:

```
# Strategy0: loop over a copy of the collection
a = [1,2,1,2,1,1,1]
for index, value in enumerate(a.copy()): # don't panic!
    if value == 1:
        a.remove(value)
print(a) # a = [2,2]
```

1. Modification when looping

Instead, it is usually more straight-forward to loop over a copy of the collection or to **create a new collection**:

```
# Strategy1:  create a new collection
a = [1,2,1,2,1,1,1]
b = []
for index, value in enumerate(a):
    if value != 1:
        b.append(value)
print(b) # b = [2,2]
```

2. Else clauses in loop

Much more similar to try's else rather than if's else: A try statement's else clause runs when no exception occurs, and a loop's else clause runs when no break occurs.

Example application scenario:

```
1  for n in range(2, 10):
2      for x in range(2, n):
3          if n % x == 0:
4              print(n, 'equals', x, '*', n//x)
5              break
6      else:
7          # loop fell through without finding a factor
8          print(n, 'is a prime number')
```

Output:

```
2 is a prime number
3 is a prime number
4 equals 2 * 2
5 is a prime number
6 equals 2 * 3
7 is a prime number
8 equals 2 * 4
9 equals 3 * 3
```

2. Else clauses in loop

Much more similar to try's else rather than if's else: A try statement's else clause runs when no exception occurs, and a loop's else clause runs when no break occurs.

See what will happen without break:

```
>>> for i in range(3):
...     print(i)
... else:
...     print("ELSE!")
...
0
1
2
ELSE!
```

```
>>> for i in []:
...     print(i)
... else:
...     print("ELSE!")
...
ELSE!
```

2. Else clauses in loop

Exercise: what is the output?

```
1 i = 0
2 while i < 3:
3     print(i, end=' ')
4     i += 1
5     if(i == 3):
6         break
7 else:
8     print("Stop at:", i, end=' ')
```

A. 0 1 2 B. 0 1 2 3 C. 0 1 2 stop at: 3 D. 0 1 2 3 stop at:3

2. Else clauses in loop

Exercise: what is the output?

```
1 i = 0
2 while i < 3:
3     print(i, end=' ')
4     i += 1
5     if(i == 3):
6         break
7 else:
8     print("Stop at:", i, end=' ')
```

A. 0 1 2 B. 0 1 2 3 C. 0 1 2 stop at: 3 D. 0 1 2 3 stop at:3

3. Pass statement

It does nothing!
But can be used when a statement is required syntactically but the program requires no action.

```
1 while True:
2     pass  # Busy-wait for keyboard interrupt (Ctrl+C)
```

Or creating minimal classes:

```
1 class MyEmptyClass:
2     pass
```

Or just being a place-holder for a function or conditional body; enable you to think abstractly!

```
1 def initlog(*args):
2     pass   # Remember to implement this!
```

4. Lambda: return a function

A normal function definition:

```
1 def incrementor(n, x):
2     return n + x
```

What if n is changed not so frequently?

4. Lambda: return a function

Try this to define a function with the help of an outer function:

```
def make_incrementor(n): # Return a function
    def incrementor(x):
        return n + x # Reference variables from the enclosing scope
    return incrementor
```

Or elegant lambda definition:

```
def make_incrementor(n): # Return a function too
    return lambda x: n + x
```

```
>>> foo = make_incrementor(233)
>>> foo(100)
333
```

4. Lambda: return a function

Exercise: What is the output?

```
1 def foo(x,*y,z):
2     print(x,y,z)
3     return lambda t: (t + x) * z
4 f1 = foo(1,2,3,4,z=5)
5 print(f1(7))
```

4. Lambda: return a function

Exercise: What is the output?

```
1 def foo(x,*y,z):
2     print(x,y,z)
3     return lambda t: (t + x) * z
4 f1 = foo(1,2,3,4,z=5)
5 print(f1(7))
```

Result:
1 (2, 3, 4) 5
40

4. Lambda: passed as an argument

- Another use of lambda is to pass a small function as an argument.
- Example: using lambda in sort(*, key=None, reverse=None)

```
>>> help(list.sort)
Help on method_descriptor:

sort(...)
    L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
```

- Key needs to be a function, e.g. str.lower, not str.lower()! Why? Try it out by yourself.

```
>>> l = ['A','b','C']
>>> l.sort()
>>> l
['A', 'C', 'b']
>>> l.sort(key=str.lower)
>>> l
['A', 'b', 'C']
```

4. Lambda: passed as an argument

- Another use of lambda is to pass a small function as an argument.
- Example: using lambda in sort(*, key=None, reverse=None)

```
>>> help(list.sort)
Help on method_descriptor:

sort(...)
    L.sort(key=None, reverse=False) -> None -- stable sort *IN PLACE*
```

- Key needs to be a function, e.g. str.lower
- Lambda is useful in this case!

```
>>> pairs = [(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four')]
>>> pairs.sort(key = lambda x: x[1])
>>> pairs
[(4, 'four'), (1, 'one'), (3, 'three'), (2, 'two')]
```

4. Lambda: passed as an argument

- Exercise: What is the output?

```
1 a = [-5,0,3,-4,-2,3,2]
2 a.sort(key = lambda x: (x<0, abs(x)))
3 print(a)
```

4. Lambda: passed as an argument

- Exercise: What is the output?

```
1 a = [-5,0,3,-4,-2,3,2]
2 a.sort(key = lambda x: (x<0, abs(x)))
3 print(a)
```

- Result: [0, 2, 3, 3, -2, -4, -5]

References

https://docs.python.org/3/tutorial/controlflow.html

https://docs.python.org/3/tutorial/datastructures.html

You can learn more by typing in the console! Keep practicing rather than purely reading.