```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

How Python works:

Namespaces

Some vocabulary

We'll use the terms "value" and "object" interchangeably.
We'll use the terms "name" and "variable" interchangeably.

A **binding** is a *runtime* pair: name ↦ value.

A **namespace** is a *runtime* collection of bindings.

At runtime, an **assignment** *binds* a name to a value.

At runtime, a **reference** *looks up* a name's value.

A name's **scope** is the region of text in which that name is valid.

```
x, y = 'a', 'b'

def f1():
    x = 1
    print(x, y)

def f2(y):
    x = 2
    print(x, y)

f1()
f2(3)
print(type(x), type(y))
print(x, y)
```

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ 

(and others)

1

global

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ 

(and others)

1

global

x ⟼ 'a'

y ⟼ 'b'

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

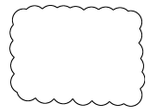
f2

scopes

(determined by program code)

2

built-in

type ⟼ 

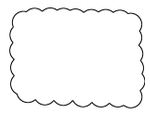
(and others)

1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ 

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ 

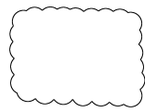
(and others)

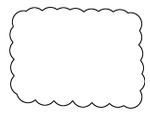
1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ 

f2 ⟼ 

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes
(determined by program code)

2

built-in

type ⟼ ▭ *(and others)*

1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ ▭

f2 ⟼ ▭

0

local

f1, called @ line 11

namespaces
(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ ▭ *(and others)*

1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ ▭

f2 ⟼ ▭

0

local

x ⟼ 1

f1, called @ line 11

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ 

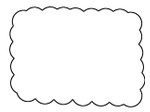
(and others)

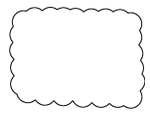
1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ 

f2 ⟼ 

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ ▭ *(and others)*

1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ ▭

f2 ⟼ ▭

0

local

y ⟼ 3

f2, called @ line 12

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ 

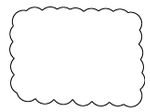
(and others)

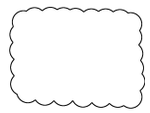
1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ 

f2 ⟼ 

0

local

y ⟼ 3

x ⟼ 2

f2, called @ line 12

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ ▭ *(and others)*

1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ ▭

f2 ⟼ ▭

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

2

built-in

type ⟼ 

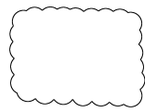
(and others)

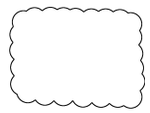
1

global

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼ 

f2 ⟼ 

namespaces

(a snapshot of program execution)

builtin

global / file / module / session

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

f1

f2

scopes

(determined by program code)

Let's practice!

```
1  def fact(n):
2    if (n == 0):
3      return 1
4    return n * fact(n-1)
5
6  n = 4
7  result = fact(n / 2)
```

Firstname Lastname T. 10 / 30

(Your response)

Modules are just more namespaces.

scopes

(determined by program code)

functions.py

```
1   x, y = 'a', 'b'
2
3   def f1():
4       x = 1
5       print(x, y)
6
7   def f2(y):
8       x = 2
9       print(x, y)
10
11  f1()
12  f2(3)
13  print(type(x), type(y))
14  print(x, y)
```

builtin

global / file / module / session

```
1  import functions
2  functions.f1()
```

scopes

(determined by program code)

2

built-in

type ⟼

1

global

functions ⟼

functions

x ⟼ 'a'

y ⟼ 'b'

f1 ⟼

f2 ⟼

namespaces

(a snapshot of program execution)

https://commons.wikimedia.org/wiki/File:Illustration_of_a_duck-billed_platypus._Wellcome_L0075037.jpg

Slicing a sequence

optional

```
seq[start : end : step]
```

think: [*start*, *end*)

```
 0   1                                   8   9
 ↓   ↓                                   ↓   ↓
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
 ↑   ↑                                   ↑   ↑
-10  -9                                  -2  -1
```

```
>>> values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> values[3:]
[3, 4, 5, 6, 7, 8, 9]
>>> values[3:5]
[3, 4]
>>> values[-1]
9
>>> values[::2]
[0, 2, 4, 6, 8]
```

Functional programming in Python

Reading from files

data.txt

```
0,1,2,3,4,5,6,7,8,9
0,2,4,6,8
100,200,300,400,500,600,700,800,900
10,17,24,31,38,45,52,59,66,73,80,87,94
```

```
open('data.txt').read()
'0,1,2,3,4,5,6,7,8,9\n0,2,4,6,8\n100,200,300,400,500,600,700,800
,900\n10,17,24,31,38,45,52,59,66,73,80,87,94\n'
```

```
open('data.txt').readlines()
['0,1,2,3,4,5,6,7,8,9\n',
 '0,2,4,6,8\n',
 '100,200,300,400,500,600,700,800,900\n',
 '10,17,24,31,38,45,52,59,66,73,80,87,94\n']
```

List comprehensions

are syntactic sugar for functional programming concepts (e.g., `map`)

```
lines = open('data.txt').readlines()
```

```
data = []
for line in lines:
  data.append(line[:-1])
```

loop
(imperative programming)

```
data = list(map(lambda line: line[:-1], lines))
```

map
(functional programming)

```
data = [line[:-1] for line in lines]
```

list comprehension
(functional programming)

List comprehensions

are syntactic sugar for functional programming concepts (e.g., `filter`)

```
positiveValues = []
for value in values:
  if value > 0:
    positiveValues.append(value)
```

loop
(imperative programming)

```
data = list(filter(lambda value: value > 0, values))
```

filter
(functional programming)

```
data = [value for value in values if value > 0]
```

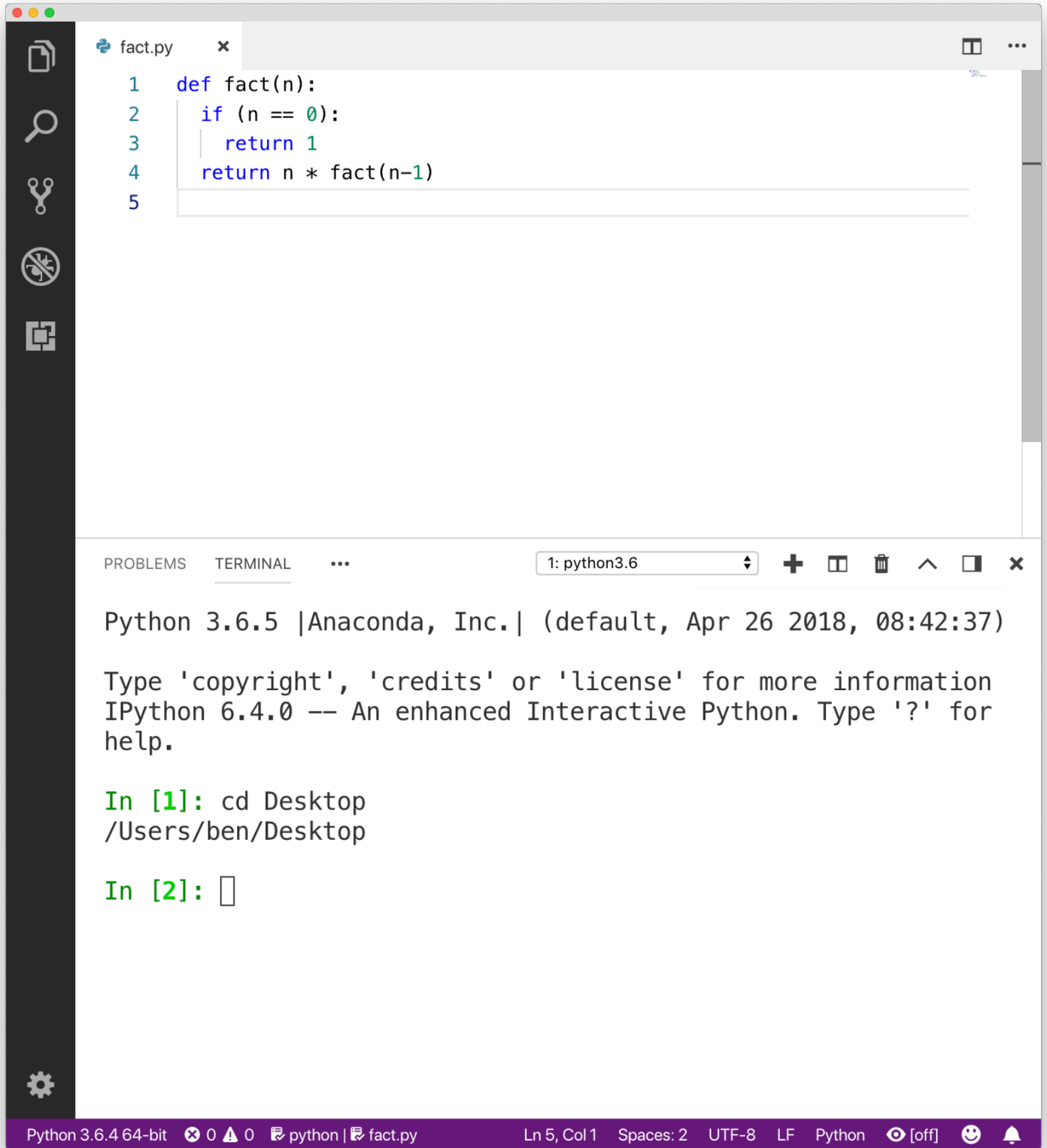
list comprehension
(functional programming)

Good programming practice

Use list comprehensions.

List comprehensions are usually clearer (to Python programmers) than `map` / `filter` or single loops that build up lists.

Python environment: VSCode + iPython



https://www.cs.hmc.edu/twiki/bin/view/CS5/Orientation

Python sounds good!

http://tinyurl.com/hmc-python-sounds

Help with the terminal: http://tinyurl.com/hmc-ipython-terminal

Try to get as far as: `replace_some`

We'll stop at 10:45.