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!
The essence of Python

Everything is an object.*

Every object has

a value.

a type.

an identity.

a namespace.

*but some objects (e.g., numeric and boolean literals) are “special”.
What kinds of things do we care about when we learn a new data structure?

- **What are dictionaries good for?**
  Looking things up

- **How do I make a dictionary?**
  literals (also: dict & comprehensions)

- **How do I query a dictionary?**
  `dictValue[key]`
  `key in dictValue`
  (also: dictionary methods)

- **How do I modify a dictionary?**
  `dictValue[key] = newValue`
  `del dictValue[key]`
  (also: dictionary methods)
def tabulate(values):
    result = {}
    for value in values:
        if value not in result:
            result[value] = 0
        result[value] += 1
    return result

Given a list of values, returns a dictionary that maps each unique value to the number of times that value occurred in the list.
def tabulate(values):
   ...

   Given a list of values, returns a dictionary that maps each unique value to the number of times that value occurred in the list ...

result = {} 

for value in values:
   
   if value not in result:
       result[value] = 0 

   result[value] += 1 

return result
from collections import defaultdict

def tabulate(values):
    """
    Given a list of values, returns a dictionary that maps each
    unique value to the number of times that value occurred in the list
    """

    result = defaultdict(int)

    for value in values:
        result[value] += 1

    return result
What does this program print, and why?

```python
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
```

(Your response)
Some vocabulary

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

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

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

At runtime, a **reference** *looks up* a variable’s value.

A **scope** is a region of text whose statements share a namespace.
```python
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
```
```python
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
```

**Scopes** (determined by program code)

**Namespaces** (a snapshot of program execution)

**Built-in**

(type and others)
```python
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)
```
```python
x, y = 'a', 'b'
def f1():
    x = 1
    print x, y
f1()
def f2(y):
    x = 2
    print x, y
f2(3)
print type(x), type(y)
print x, y
```

**Scopes** (determined by program code)

**Namespaces** (a snapshot of program execution)

**Built-in**
- `type` (and others)

**Global**
- `x` → 'a'
- `y` → 'b'
- `f1` → 

**Scopes**

**File / Module / Session**

**Namespaces**

**Built-in**

**Global**

**Scopes**

**File / Module / Session**

**Namespaces**

**Built-in**

**Global**

**Scopes**

**File / Module / Session**

**Namespaces**

**Built-in**

**Global**
```
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
```
```python
x, y = 'a', 'b'
def f1():
    x = 1
    print x, y
f1()
def f2(y):
    x = 2
    print x, y
f2(3)
print type(x), type(y)
print x, y
```

**scopes**
(determined by program code)

**namespaces**
(a snapshot of program execution)

**built-in**
(type (and others))

**global**

- x → 'a'
- y → 'b'
- f1 →
- f2 →

**local**

- f1, called @ line 11
```python
file / module / session

x, y = 'a', 'b'

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

    f1()
    f2(3)

    print type(x), type(y)

print x, y
```

### scopes
(determined by program code)

### namespaces
(a snapshot of program execution)

### built-in
(type and others)

```
1  x ➞ 'a'  
  y ➞ 'b'
  f1 ➞    
  f2 ➞    

0  local
  x ➞ 1

f1, called @ line 11
```
```python
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
```
```python
x, y = 'a', 'b'

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

def f2(y):
    x = 2
    print x, y
f2(3)

print type(x), type(y)
print x, y
```

**Scopes**
(determined by program code)

**Namespaces**
(a snapshot of program execution)
```python
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
```
```python
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)
```

**Scopes**
(determined by program code)

**Namespaces**
(a snapshot of program execution)
```python
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
```

**scopes**  
(determined by program code)

**namespaces**  
(a snapshot of program execution)
```python
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)
```

**scopes**
(determined by program code)