>>> 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!
Fill out the use-it-or-lose-it template for min-dist

Firstname Lastname

(Your response)
min-dist?: use it or lose it

a recursive strategy — fill in the pieces

Base case(s)
the min-dist between a node and itself is 0
the min-dist between two different nodes in an empty graph is ∞

Recursive case(s)
1. **It**: a piece of the problem
   an edge, $X \rightarrow Y$, with weight $w$

2. **Smaller version**: What does the input look like without it?
   the graph without that edge, called “sub-graph”

3. **Lose it**: How could we solve a smaller version *without* it?
   what is the min-dist from $n_1$ to $n_2$ in sub-graph?

4. **Use it**: How could we solve a smaller version *with* it?
   what is the min-dist from $n_1$ to $X$ in sub-graph PLUS
   the min-dist from $Y$ to $n_2$ in sub-graph PLUS $w$?

5. How would we **combine** the solutions to solve the full problem?
   the minimum of use-it and lose-it
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 \( \mapsto \) 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)
```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)