Contrasting 'import module' vs 'from module import function' in Python

Python provides two main ways to bring code from one module into another: using 'import module' and 'from module import function'. They look similar but behave quite differently in how they create references.

1. Using 'import module'

When you use 'import module', Python loads the module (if it’s not already loaded) and binds a reference to the module object in your current namespace. You then access its contents through the module name.

```
# math_example.py
import math

print(math.pi)          # access attribute through module name
print(math.sqrt(16))    # call a function through module reference

math.pi = 3.1416        # modifying math.pi changes it for everyone who
imported math
```

In this form, 'math' is a reference to the entire module object. All attributes (functions, variables, classes) remain inside that object, and you always access them using the module name as a prefix.

2. Using 'from module import function'

When you use 'from module import function', Python copies the *reference* to that specific function (or variable) into your local namespace. You can call it directly without the module prefix, but it’s no longer linked to the module’s name.

```
# math_example.py
from math import sqrt, pi

print(pi)        # direct access, no module name needed
print(sqrt(16))  # direct access

pi = 3.14        # changes only the local name 'pi' in this file, not
math.pi
```

Here, 'pi' and 'sqrt' are local references copied from the 'math' module’s namespace. If you later reassign 'pi' in your file, it won’t affect 'math.pi' or anyone else who imported math.

3. Analogy — A Mutable List

Think of a module as a mutable list. When you 'import module', you create a reference to the entire list. When you 'from module import function', you copy a reference to one element from the list. If you change an element through the full reference, everyone sees it. But if you change your own local copy, the list itself — and others who reference it — do not.

```
# analogy
myList = [10, 20, 30]

# 'import module' analogy
ref = myList                # ref points to the same list object
ref[0] = 99                 # modify element through ref
print(myList)               # [99, 20, 30]  <-- both see the change

# 'from module import function' analogy
ref = myList[0]             # ref gets a copy of the element
ref = 77                    # rebinds local ref; doesn't affect myList
print(ref)                  # 77  <-- local change only
print(myList)               # [99, 20, 30]  <-- list unaffected
```

In the analogy:
- 'import module' ≈ assigning a reference to the entire list (shared object).
- 'from module import function' ≈ copying one element's reference — rebinding it doesn't change the list.

4. Summary Comparison

Form	What It Imports	Access Style	Linked to Module?	Analogy
import module	The entire module object	module.function()	Yes – shared reference	Like ref = myList
from module import function	A specific object (function, variable, etc.)	function()	No – separate local name	Like ref = myList[0]

5. Key Takeaways

• ‘import module’ brings in a reference to the module object; any changes to its contents are seen by all importers.

• ‘from module import function’ copies a reference to a specific name — later changes to the module don’t automatically update your local copy.

• Conceptually, importing a module is like referencing a list, while importing a function is like referencing one element inside that list — changing your local element doesn’t affect the list.