In Python, **circular dependencies** happen when two or more modules depend on each other, creating an import loop.
The difference between **direct** and **indirect** circular dependencies is about *how* that loop is formed.

1. Direct Circular Dependency

- **Definition**: Two modules import each other **directly**.
- **Example**:

```
# file: a.py
import b

def func_a():
    print("Function A")

# file: b.py
import a

def func_b():
    print("Function B")
```

Flow:

- Python loads `a.py` → sees `import b` → starts loading `b.py`.
- `b.py` imports `a` again → but `a` is not fully loaded yet → leads to `ImportError` or partially initialized module.

Key point: The loop is **one step** — `a` imports `b`, and `b` imports `a`.

2. Indirect Circular Dependency

- **Definition**: The circular import happens **through one or more intermediate modules**.
- **Example**:

```
# file: a.py
import b

# file: b.py
```

```
import c

# file: c.py
import a
```

Flow:

- Python loads `a.py` → imports `b.py`.
- `b.py` imports `c.py`.
- `c.py` imports `a.py` again → but `a` is still loading → circular dependency occurs.

Key point: The loop is **multi-step** — `a → b → c → a`.

Summary Table

Type	Modules Involved	Import Path Example	Loop Length
Direct	2	`a → b → a`	2
Indirect	3 or more	`a → b → c → a`	≥3

✅ **Best Practices to Avoid Circular Imports**:

- Move imports **inside functions** where needed.
- Refactor shared code into a **third module**.
- Use **dependency injection** instead of importing.
- Avoid **tight coupling** between modules.

If you want, I can give you a **visual diagram** showing direct vs indirect circular dependencies in Python so it's easier to remember.
Do you want me to prepare that?