How Python Generator Work

Python functions work

Frame Object

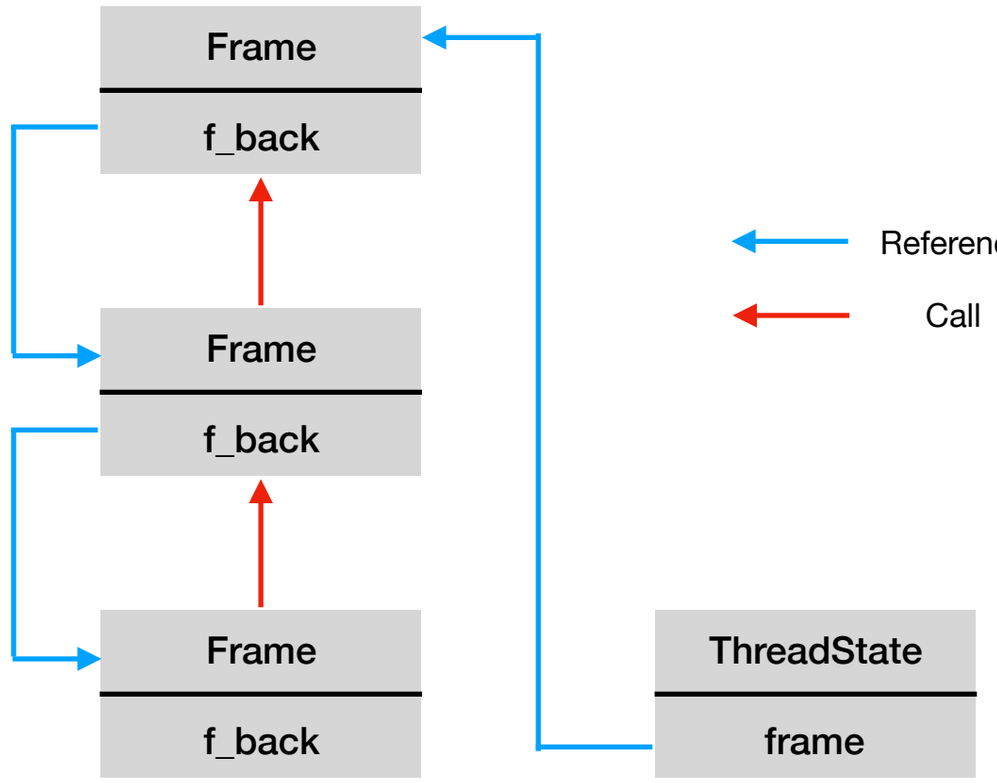
- Contains information for executing a function
- f_locals, f_lasti, f_back, f_code …

f_locals : Local variables stored (지역 변수 상태)

f_lasti : Index of last attempted instruction in bytecode

f_back : Previous stack frame, this frame's caller

f_code : Code object
- co_names : Global variable names
- co_varnames : Local variable names
- co_code : Complied bytecode
- co_consts : Constant variable



인터프린터 내부에는 각 스레드마다 ThreadState 라는 객체가 있다.
현재 실행되고 있는 함수를 멤버로서 가지고 있다.
함수가 다른 함수를 호출하면 Frame 객체가 생성되고
새롭게 생긴 프레임의 f_back 은 호출한 프레임을 가르킨다.
ThreadState 는 현재 실행중인 프레임을 갱신하게 된다.

Python functions work

```
>>> def foo():
...     bar()
...
>>> def bar():
...     pass
```

```
>>> import dis
>>> dis.dis(foo)
  2           0 LOAD_GLOBAL              0 (bar)
              2 CALL_FUNCTION            0
              4 POP_TOP
              6 LOAD_CONST               0 (None)
              8 RETURN_VALUE
```

파이썬 함수가 다른 함수(서브루틴)을 호출하면
서브루틴은 반환하기 전까지 제어권을 가지고 있다.
반환되면 제어권은 다시 호출자(caller)가 가져온다.

PyEval_EvalFrameEx 가 CALL_FUNCTION 바이트 코드를 만나면 새로운 Python 스택 프레임을 생성한다.

PyEval_EvalFrameEx 란?
프레임을 실행하는 함수. 바이트 코드들의 opcode 가 여기서 실행된다.

Python functions work

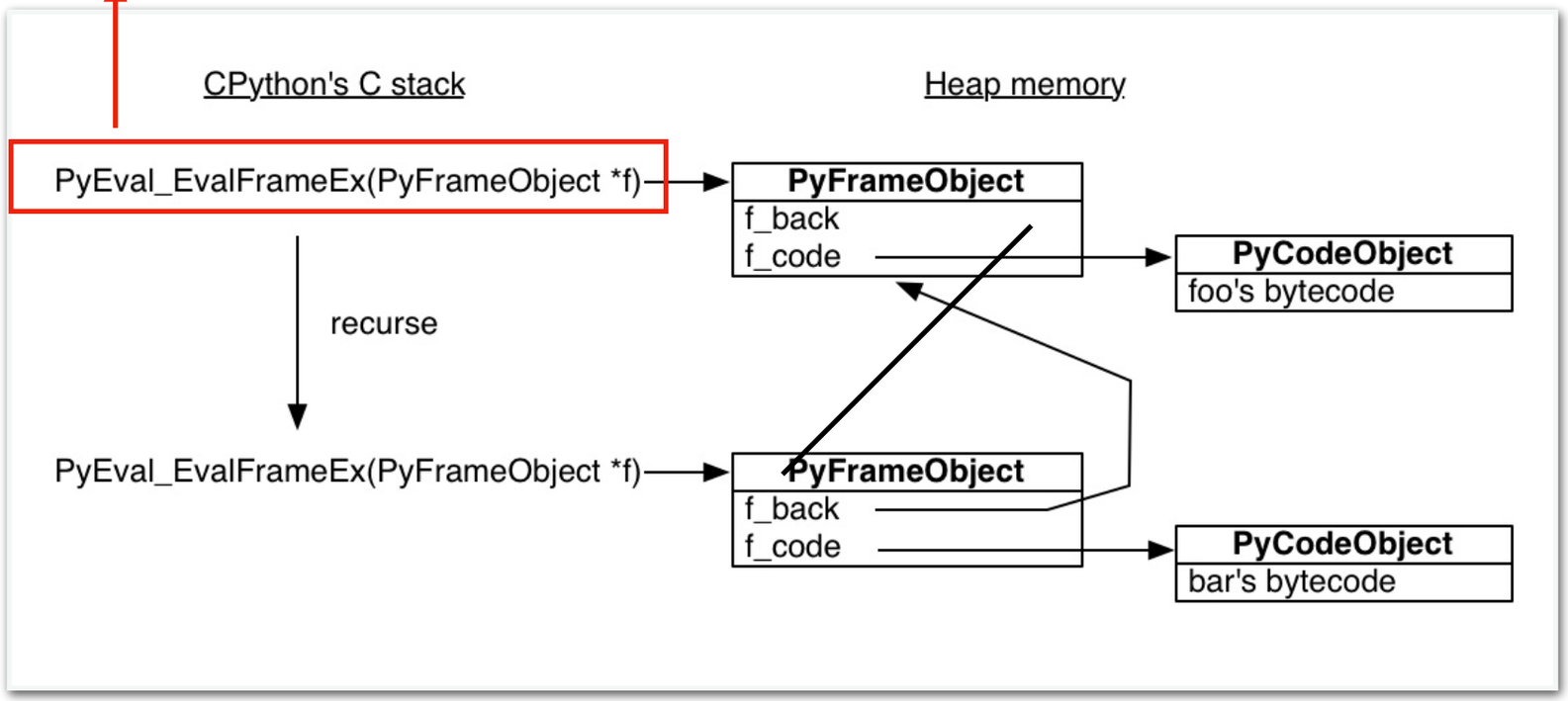
inspect 모듈은 modules, classes, methods, functions, tracebacks, frame objects, and code objects 에 대한 정보를 얻는 데 여러 가지 유용한 함수를 제공

```
>>> import inspect
>>> frame = None
>>> def foo():
...     bar()
...
>>> def bar():
...     global frame
...     frame = inspect.currentframe()
...
>>> foo()
>>> frame.f_code.co_name
'bar'
>>> caller_frame = frame.f_back
>>> caller_frame.f_code.co_name
'foo'
```

https://docs.python.org/3/library/inspect.html

Python functions work

Python 함수를 실행하는 C 함수.
파이썬 스택 프레임 객체를 취한다.



Python generators work

```
>>> def gen_fn():
...     result = yield 1
...     print('result of yield: {}'.format(result))
...     result2 = yield 2
...     print('result of 2nd yield: {}'.format(result2))
...     return 'done'
```

파이썬은 gen_fn 을 바이트 코드로 컴파일 할 때 yield 문을 보고 gen_fn 이 일반함수가 아니라 generator 함수라고 판단한다. 그리고 이 사실을 기억하는 플래그를 설정한다.

Python generators work

```
>>> generaotr_bit = 1 << 5
>>> bool(gen_fn.__code__.co_flags & generaotr_bit)
True
```

generator 함수를 호출하면 파이썬은
generator 플래그를 보고 실제로
함수를 실행하지 않고 generator 를 생성한다.

파이썬 generator 는
스택 프레임과 gen_fn의 본문을
참조하고 있는 코드를 캡슐화한다.

```
>>> gen = gen_fn()
>>> type(gen)
<class 'generator'>
>>> gen.gi_code.co_name
'gen_fn'
```

Python generators work

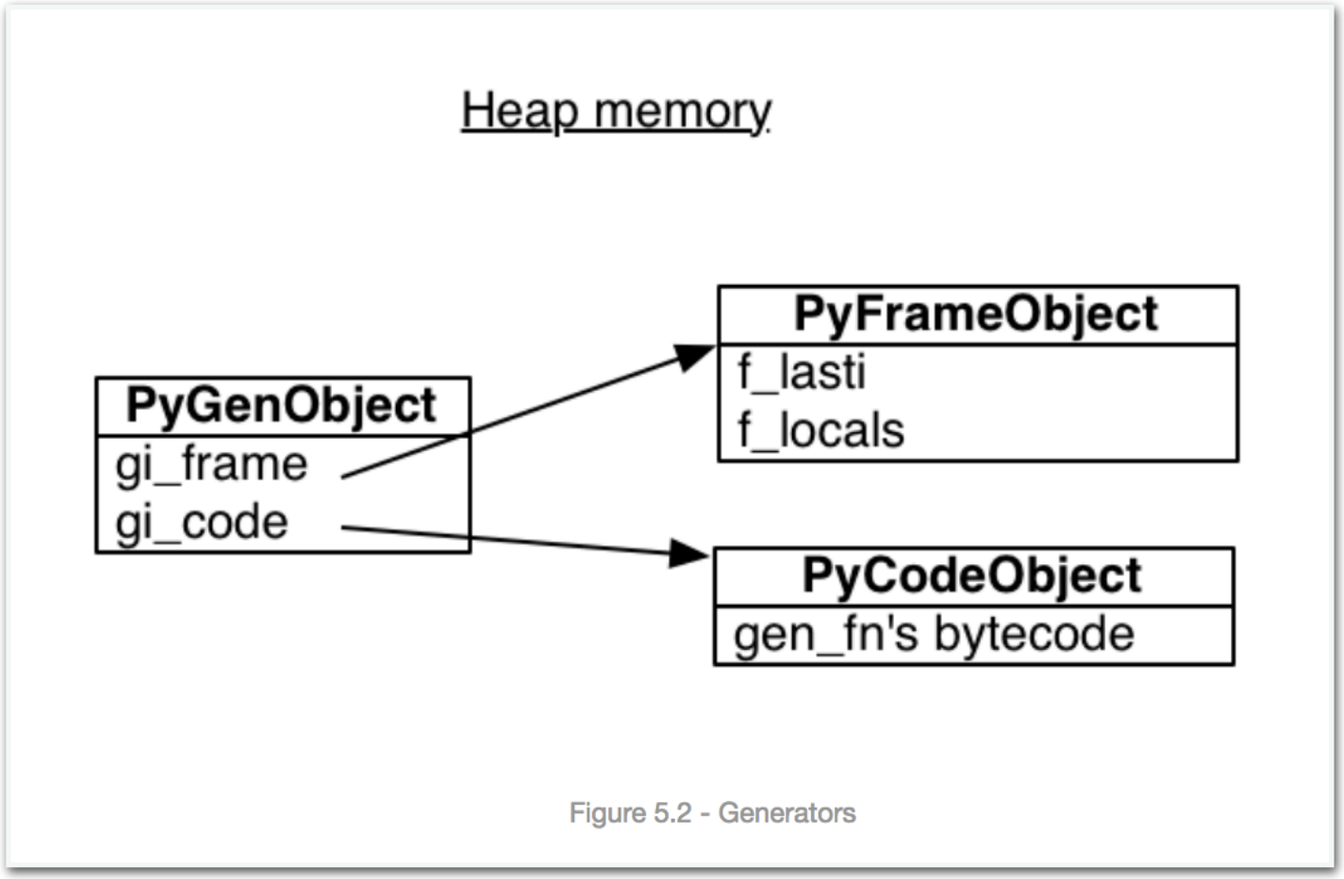


Figure 5.2 - Generators

gen_fn 을 호출 한 모든 generator 는 동일한 코드를 가르킨다.
그러나 각 generator 에는 자신들만의 구조가있다.
이 스택 프레임은 실제 스택에 없으며 힙 메모리에 있다.

Python generators work

```
>>> gen.gi_frame.f_lasti
-1
>>> gen.send(None)
1
>>> gen.gi_frame.f_lasti
2
>>> len(gen.gi_code.co_code)
44
```

프레임에는 가장 최근에 실행된 instruction 인 “last instruction” 포인터가 있다.
처음에는 last instruction 포인터가 -1 인데 이는 generator 가 시작되지 않았음을 의미.

Python generators work

generator 는 스택 프레임이 실제로
스택에 없기 때문에 (힙에 있다)
언제든지 다시 시작할 수 있다.

호출 계층 구조에서 고정되어 있지 않으며
일반 함수가 수행하는 실행 순서를
따르지 않아도된다.

```
>>> gen.send('hello')
result of yield: hello
2
```

```
>>> gen.gi_frame.f_locals
{'result': 'hello'}
```

```
>>> gen.send('goodbye')
Traceback (most recent call last):
result of 2nd yield: goodbye
  File "<input>", line 1, in <module>
StopIteration: done
```

Conclusion

Frame object

- 함수가 실행될때 사용된다.
- 함수가 실행하는데 필요한 정보들을 담고 있다.
    - Call stack
    - Value stack
    - Local varaibles
    - Last attempted bytecode instruction

Generator

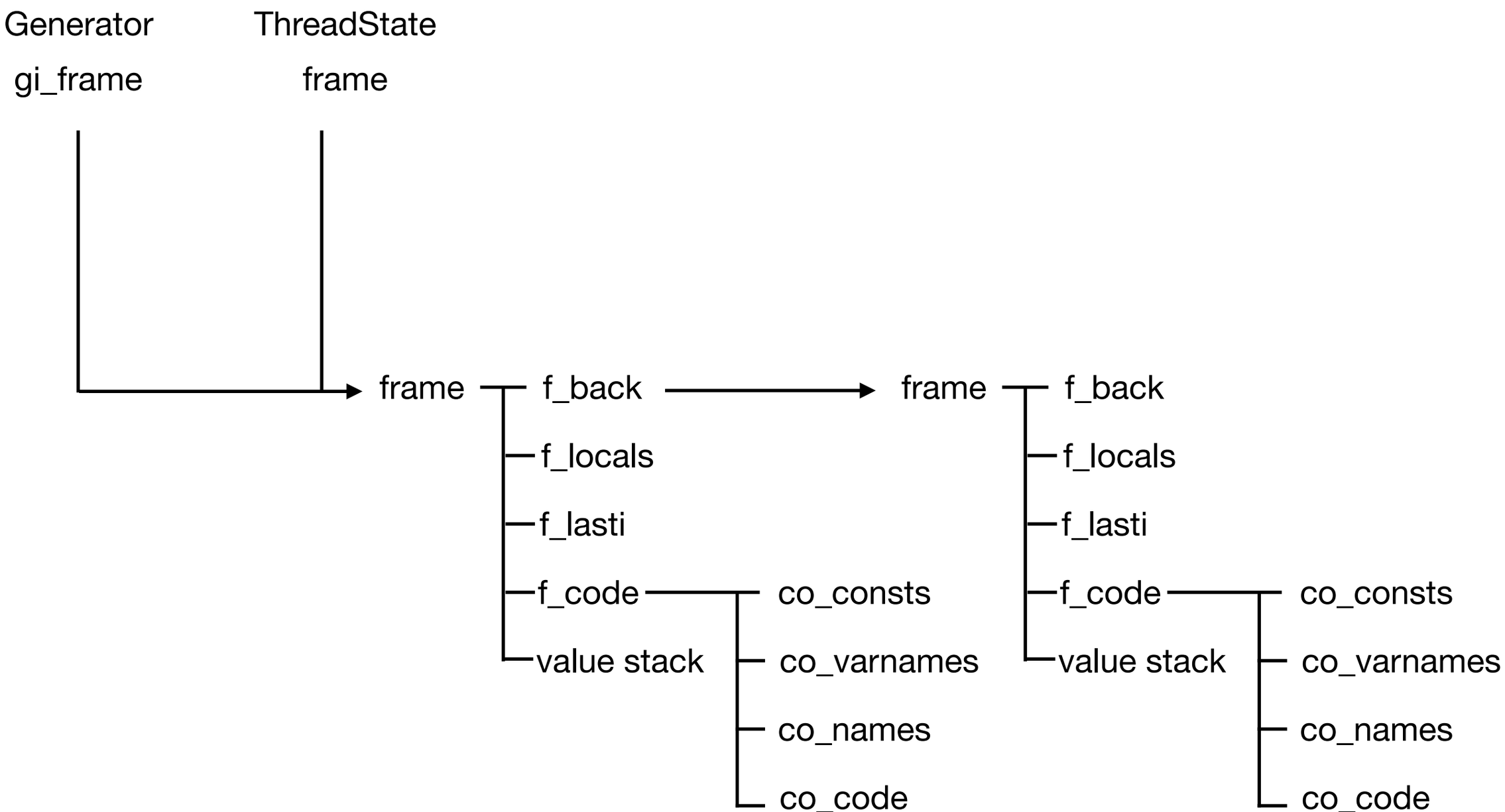
- Contains a frame object like thread state
- The frame memorizes which index of bytecode is executed.
- The frame stores local variables.

어디까지 진행되었는지 generator 가 프레임을 들고있음으로서 알 수 있기 때문에 일시중지한 다음에 다시 시작할 수 있다.

Reference

https://www.youtube.com/watch?v=NmSeLspQoAA&t=940s
https://lewissbaker.github.io/2017/09/25/coroutine-theory
https://medium.com/@pekelny/weird-python-coroutines-643c7a70f14c
https://www.geeksforgeeks.org/coroutine-in-python/amp/

Conclusion


Generator
gi_frame
ThreadState
frame
frame
f_back
f_locals
f_lasti
f_code
value stack
co_consts
co_varnames
co_names
co_code
frame
f_back
f_locals
f_lasti
f_code
value stack
co_consts
co_varnames
co_names
co_code