FUNCTIONS:

GENERATORS

Overview:

- distinguish *return* and *yield* statements
- learn the role of generators

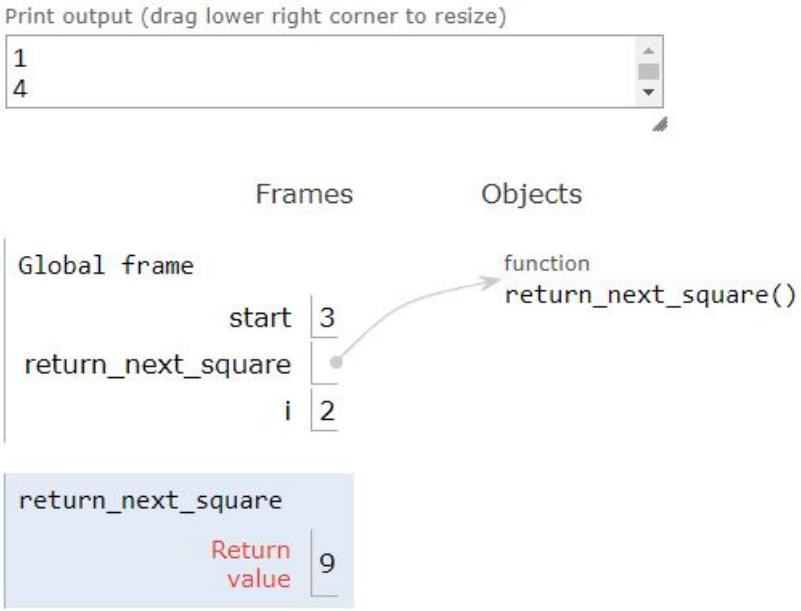
return **vs.** *yield*

- *return*:
  1. a value to a caller
- *yield*:
  1. sequence of values to a caller
  2. sequence is not stored
  3. suspends function
  4. retains state to resume
  5. used in iteration

return Example

```
start = 0
def return_next_square():
    global start
    start = start + 1
    return start * start

for i in range(5):
    print(return_next_square())
```



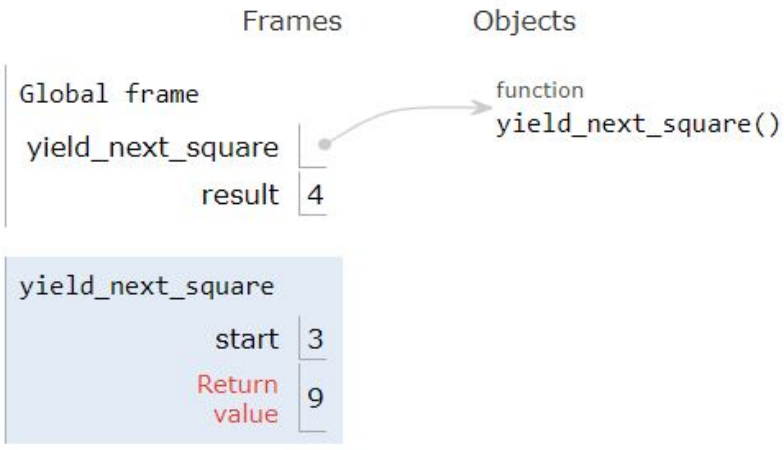
yield Example

```
def yield_next_square():
    start = 1
    while True:
        yield start * start
        start = start + 1

for result in yield_next_square():
    print(result)
    if result > 25:
        break
```

Print output (drag lower right corner to resize)

```
1
4
```



Generators

- many objects are iterable
  1. use *for* in iteration
  2. *__iter__*() method returns an iterator
  3. iterators have *__next__*() method
- generators - functions that return *iterable* objects
  1. implement *__iter__*() & *__next__*()
  2. computation "on-demand"

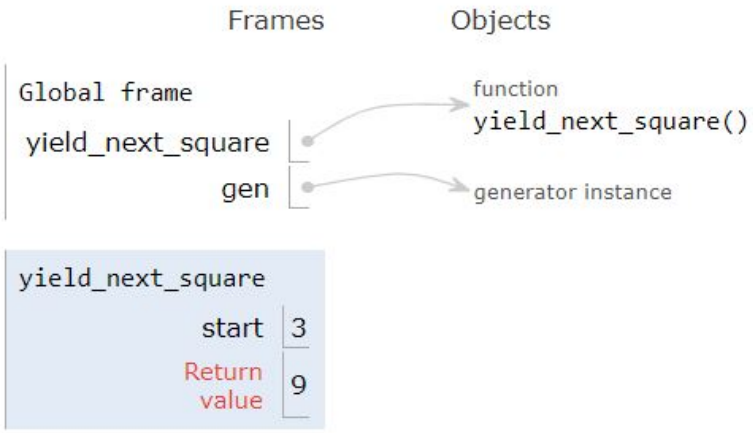
next in Generators

```
def yield_next_square():
    start = 1
    while True:
        yield start * start
        start = start + 1

gen = yield_next_square()
print(next(gen))
print(next(gen))
print(next(gen))
```

Print output (drag lower right corner to resize)

```
1
4
```



Exercise(s):

- write function $next\_arith()$ to produce the next value in arithmetic progression $A(a, d)$

  $a, a+d, a+2d, \ldots, a+nd, \ldots$

- write function $next\_geom()$ to produce next value in geometric progression $G(b, q)$

  $b, bq, bq^2, \ldots, bq^n, \ldots$