Machine Learning: Optional: Advanced Python

CPSC 501: Advanced Programming Techniques
Winter 2025

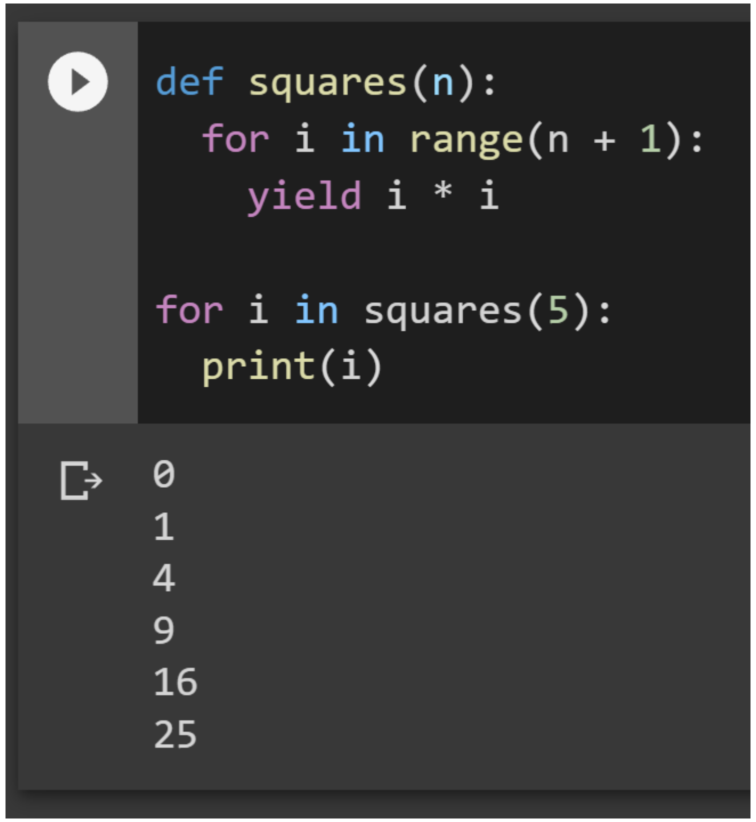
Jonathan Hudson, Ph.D
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

Thursday, February 13, 2025

UNIVERSITY OF CALGARY

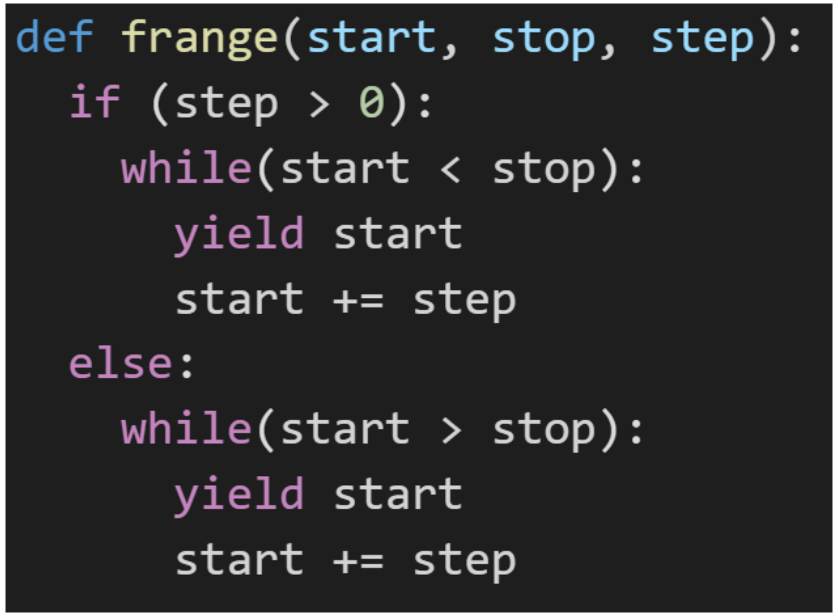
Generators

- Function that returns on each yield call, tracking state so it can be restarted to continue



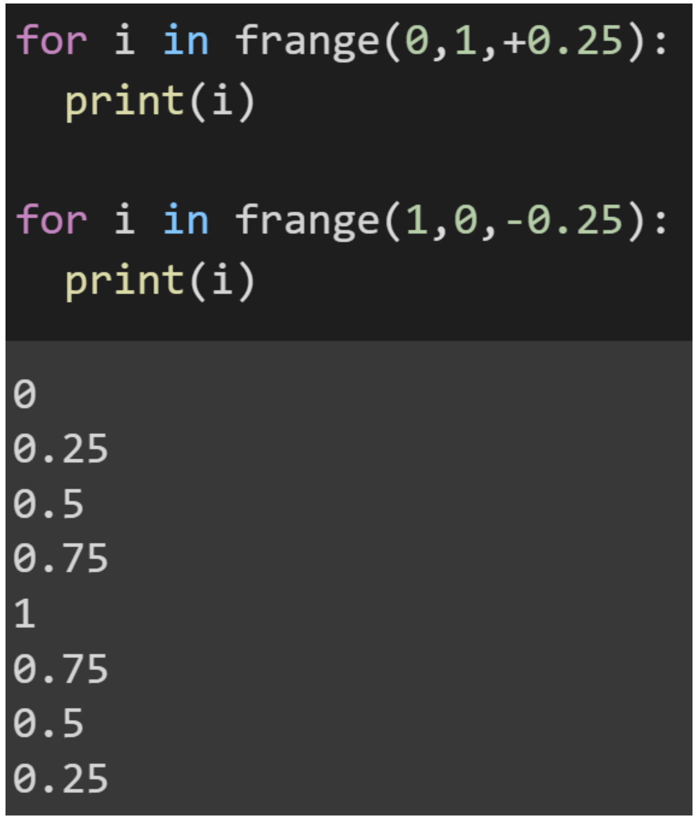
```
def squares(n):
  for i in range(n + 1):
    yield i * i

for i in squares(5):
  print(i)
```

```
0
1
4
9
16
25
```



```
def frange(start, stop, step):
  if (step > 0):
    while(start < stop):
      yield start
      start += step
  else:
    while(start > stop):
      yield start
      start += step
```

Can make a quick float range function this way for looping



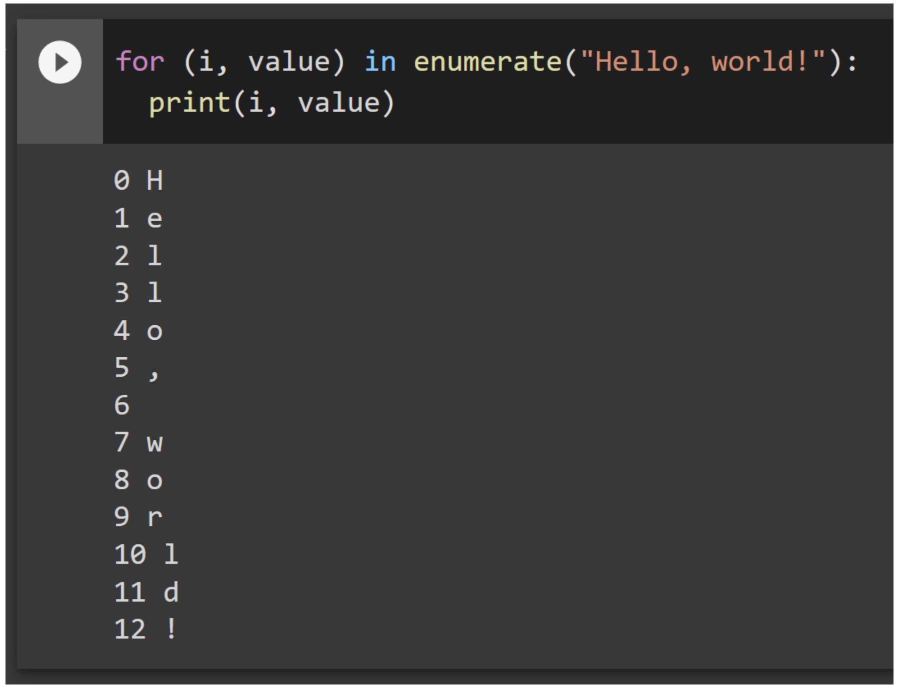
```
for i in frange(0,1,+0.25):
  print(i)

for i in frange(1,0,-0.25):
  print(i)
```

```
0
0.25
0.5
0.75
1
0.75
0.5
0.25
```

Enumerate

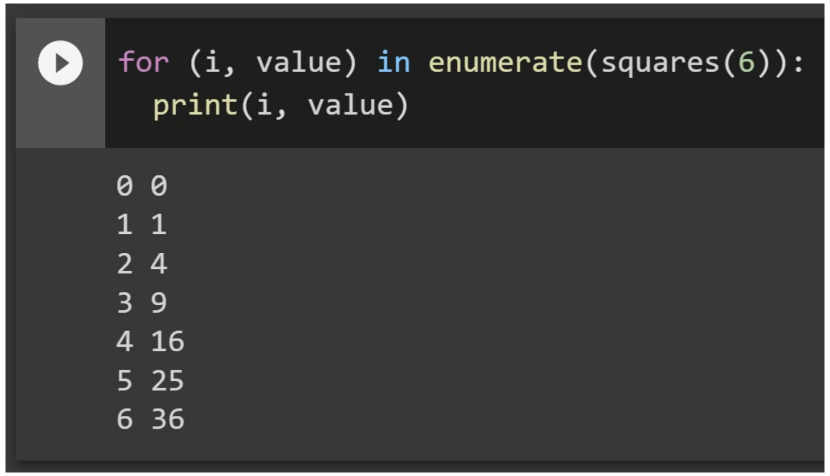
Creates tuple with incrementing index

Input can be any generator



```
for (i, value) in enumerate("Hello, world!"):
  print(i, value)
```

```
0 H
1 e
2 l
3 l
4 o
5 ,
6  
7 w
8 o
9 r
10 l
11 d
12 !
```

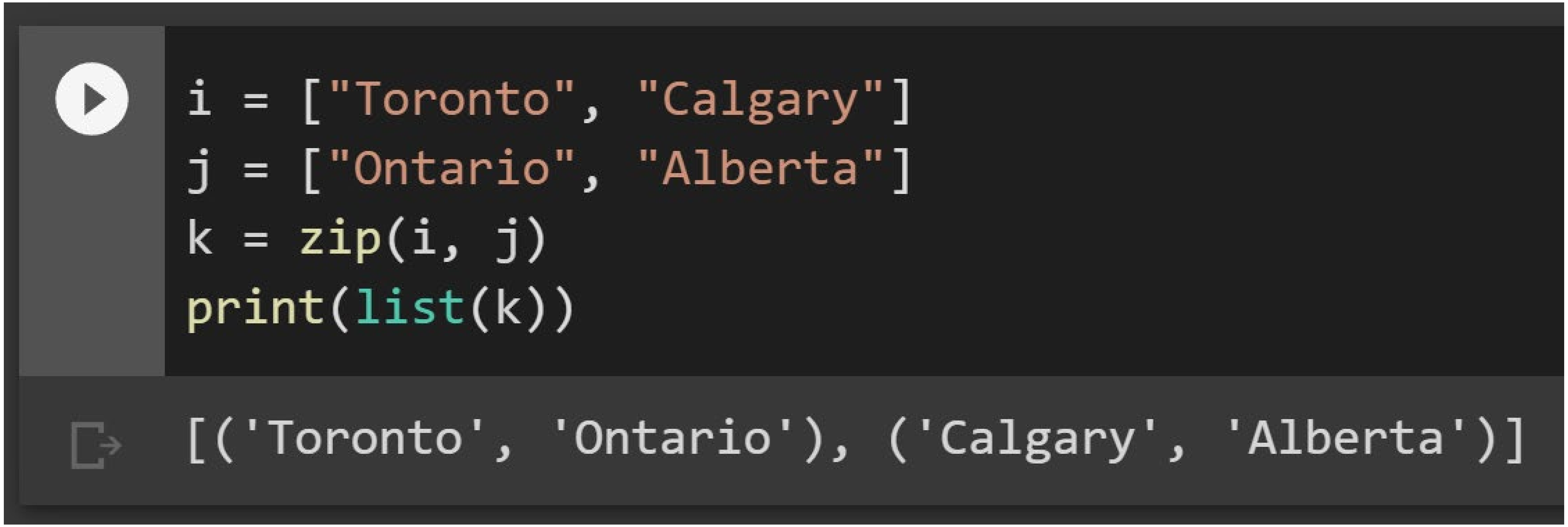


```
for (i, value) in enumerate(squares(6)):
  print(i, value)
```

```
0 0
1 1
2 4
3 9
4 16
5 25
6 36
```

Zip

- Put two similar length things together (list/tuples)
- Extra length part in either list would be ignored

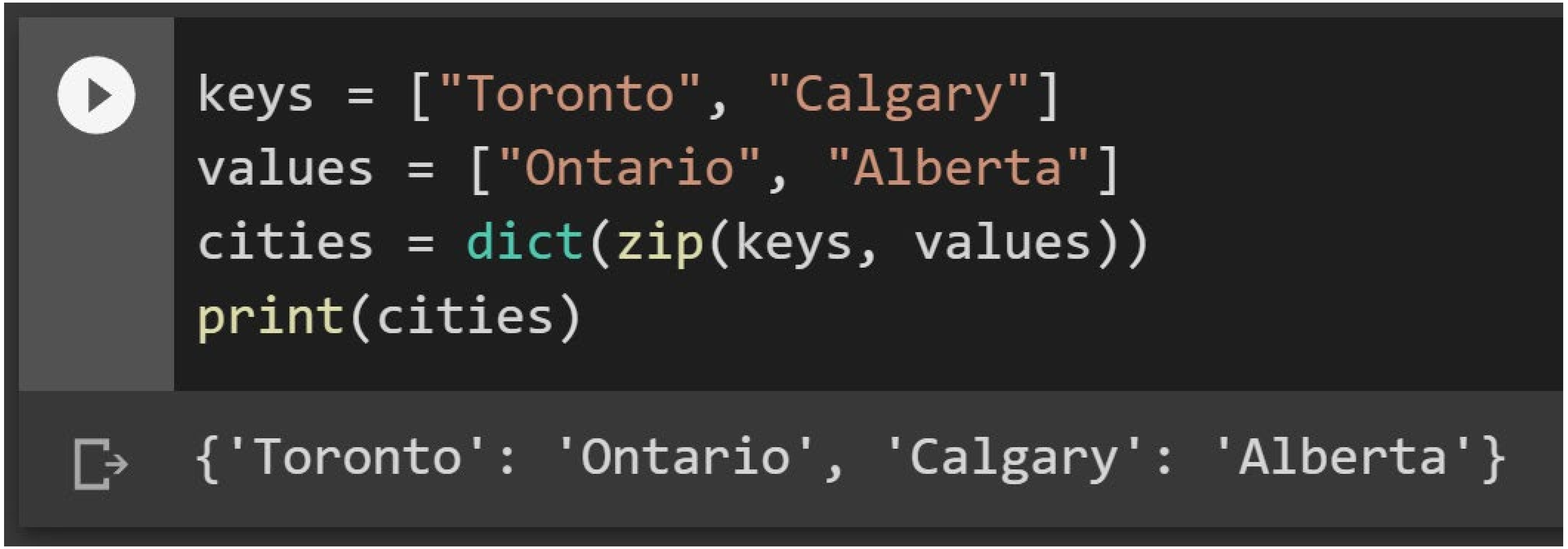


```
i = ["Toronto", "Calgary"]
j = ["Ontario", "Alberta"]
k = zip(i, j)
print(list(k))
```

```
[('Toronto', 'Ontario'), ('Calgary', 'Alberta')]
```

Zip to dictionary

- Dict via zip mapping

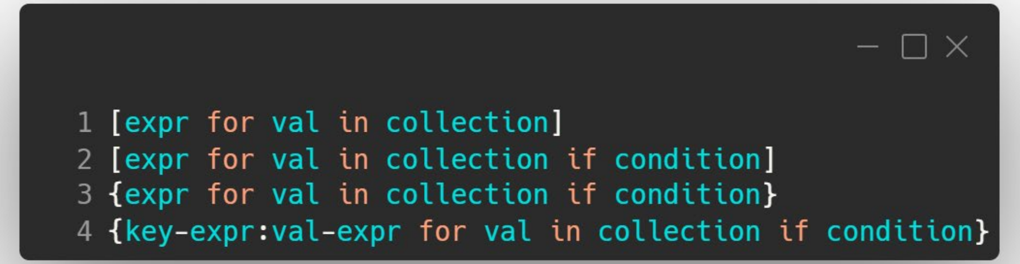


```
keys = ["Toronto", "Calgary"]
values = ["Ontario", "Alberta"]
cities = dict(zip(keys, values))
print(cities)
```

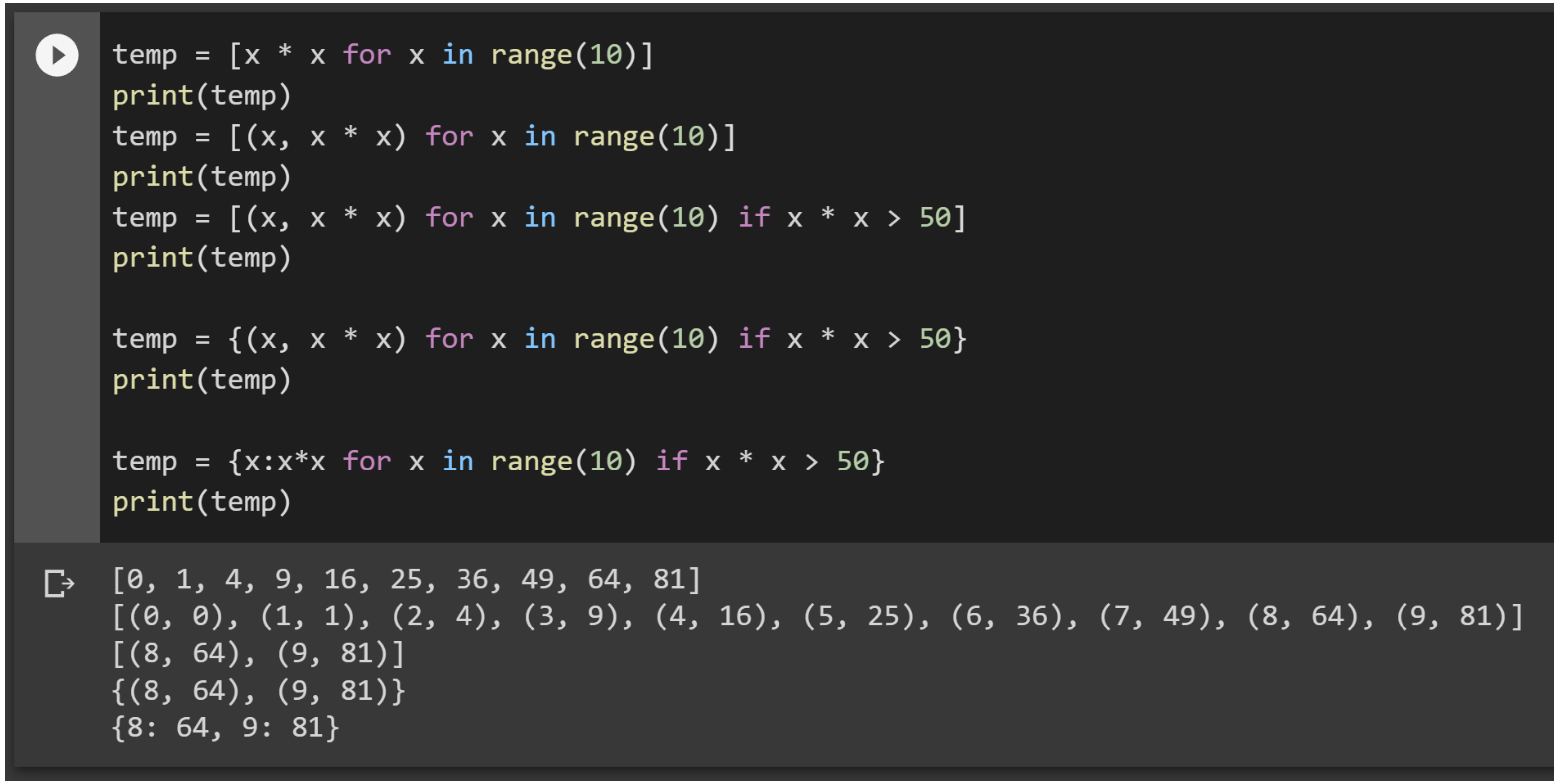
```
{'Toronto': 'Ontario', 'Calgary': 'Alberta'}
```

Data structure comprehension

- Create list in-line
- dict/set applicable



```
[expr for val in collection]
[expr for val in collection if condition]
{expr for val in collection if condition}
{key-expr:val-expr for val in collection if condition}
```

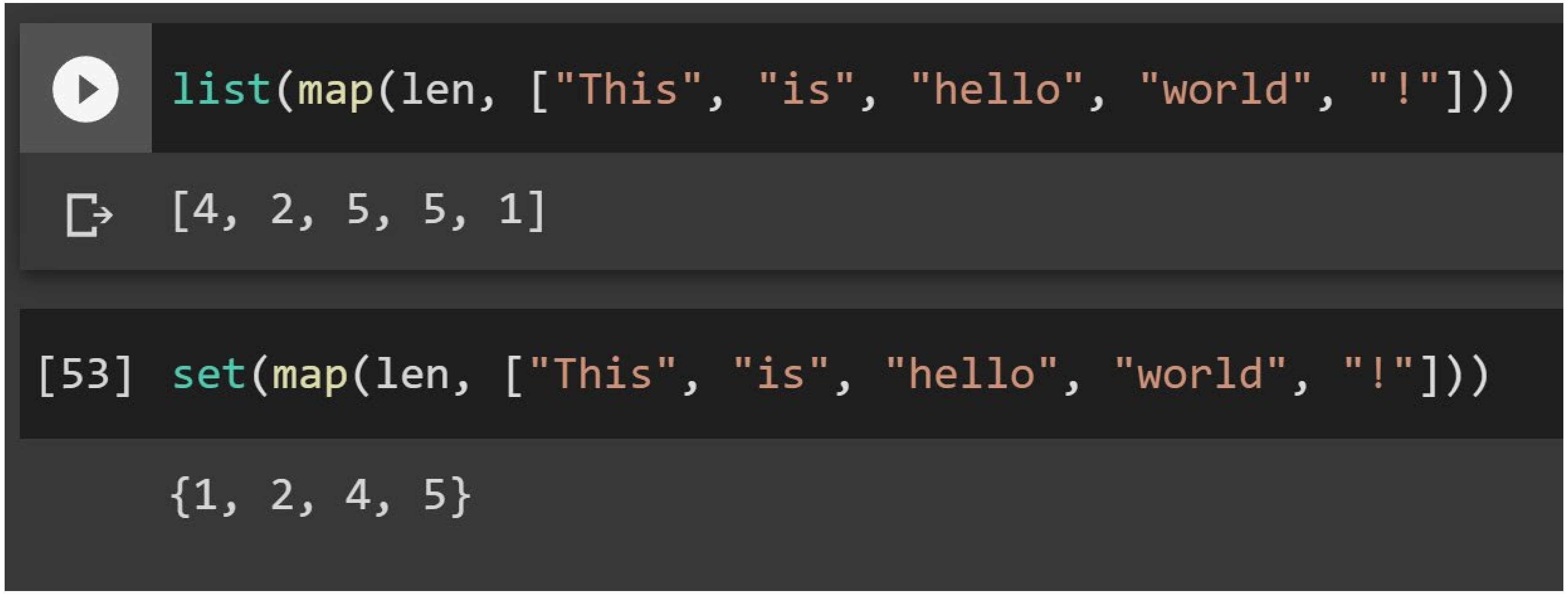


```
temp = [x * x for x in range(10)]
print(temp)
temp = [(x, x * x) for x in range(10)]
print(temp)
temp = [(x, x * x) for x in range(10) if x * x > 50]
print(temp)

temp = {(x, x * x) for x in range(10) if x * x > 50}
print(temp)

temp = {x:x*x for x in range(10) if x * x > 50}
print(temp)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
[(8, 64), (9, 81)]
{(8, 64), (9, 81)}
{8: 64, 9: 81}
```

Mapping

- Map a function onto another collection

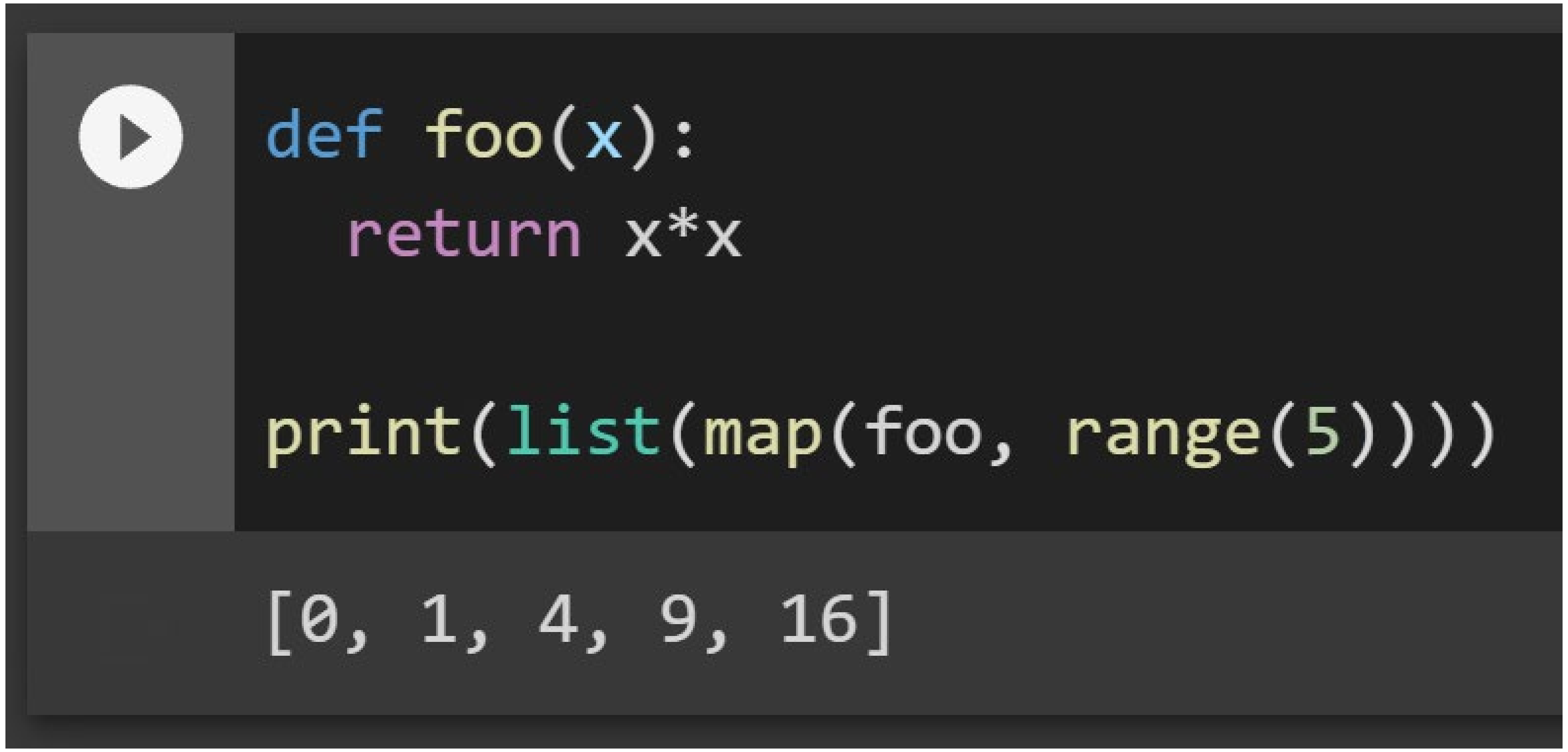


```
list(map(len, ["This", "is", "hello", "world", "!"]))
[4, 2, 5, 5, 1]
```

```
[53] set(map(len, ["This", "is", "hello", "world", "!"]))
{1, 2, 4, 5}
```

Mapping – functions are objects

- Even your own function can be applied as mapping

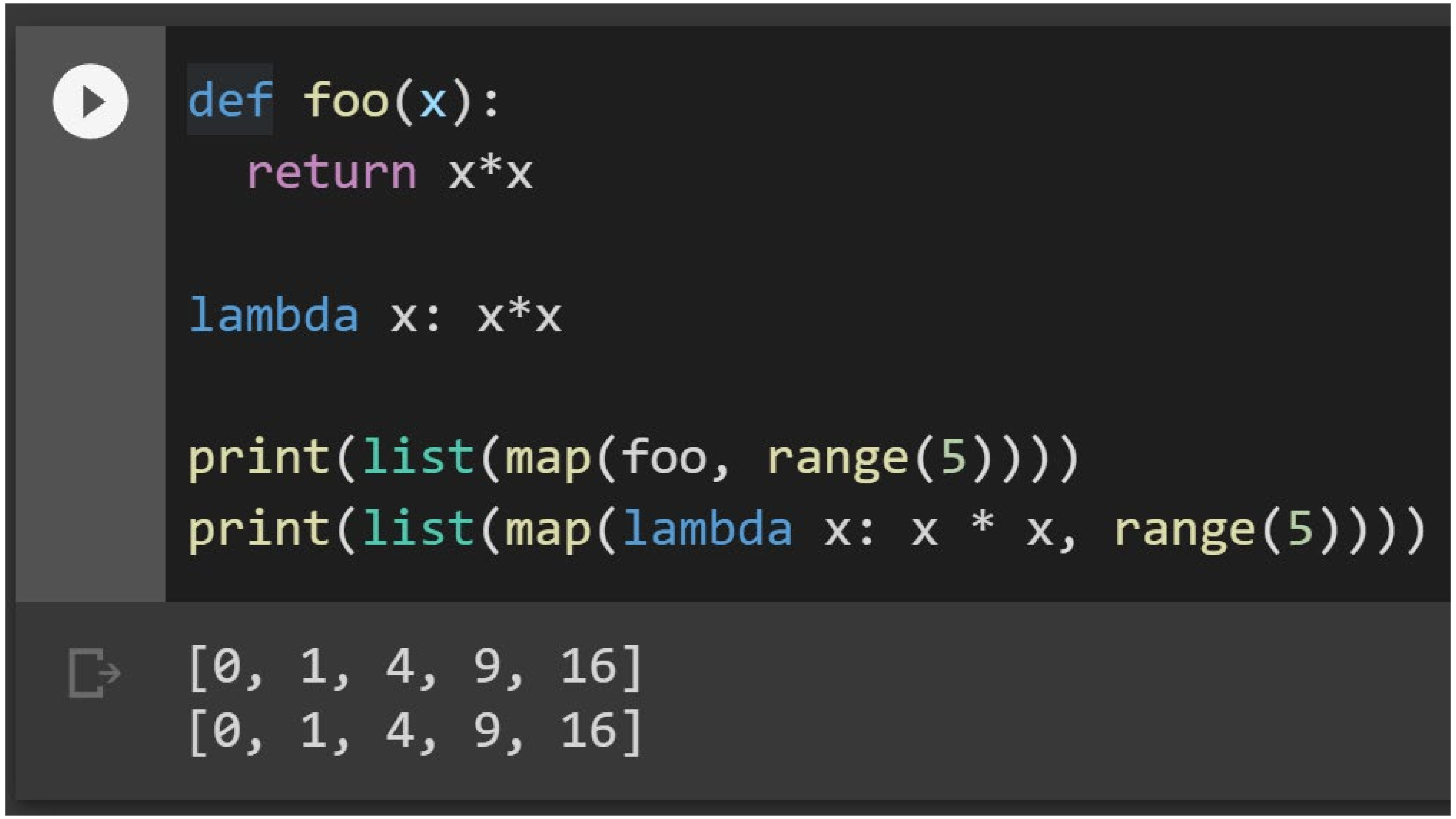


```python
def foo(x):
  return x*x

print(list(map(foo, range(5))))
```

```
[0, 1, 4, 9, 16]
```

Lambdas

- Lambda – make short function into inline expression (easy for mapping)

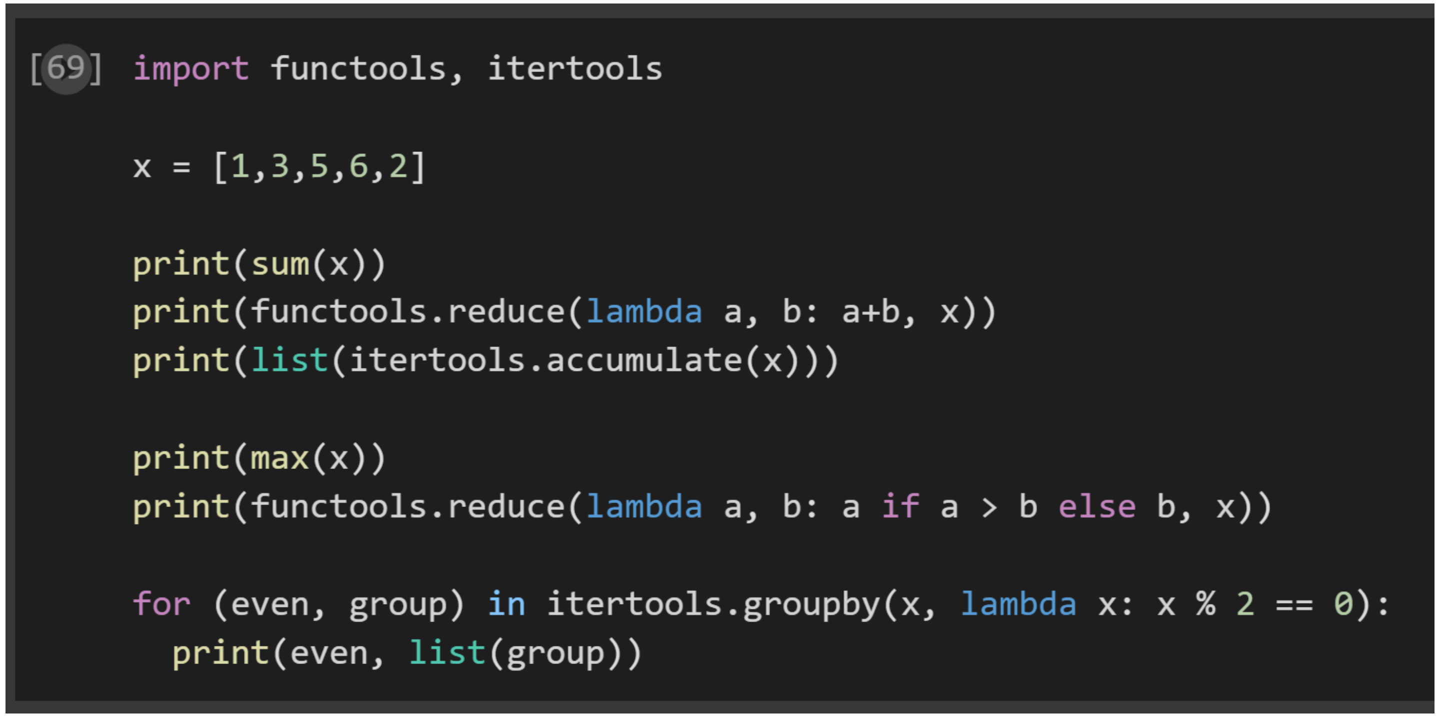


```
def foo(x):
  return x*x

lambda x: x*x

print(list(map(foo, range(5))))
print(list(map(lambda x: x * x, range(5))))
```

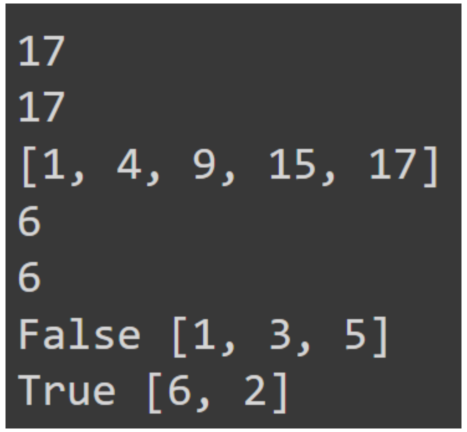
```
[0, 1, 4, 9, 16]
[0, 1, 4, 9, 16]
```

Accumulate, Reduce, Groupby

- https://docs.python.org/3/library/itertools.html accumulate, groupby
- https://docs.python.org/3/library/functools.html reduce



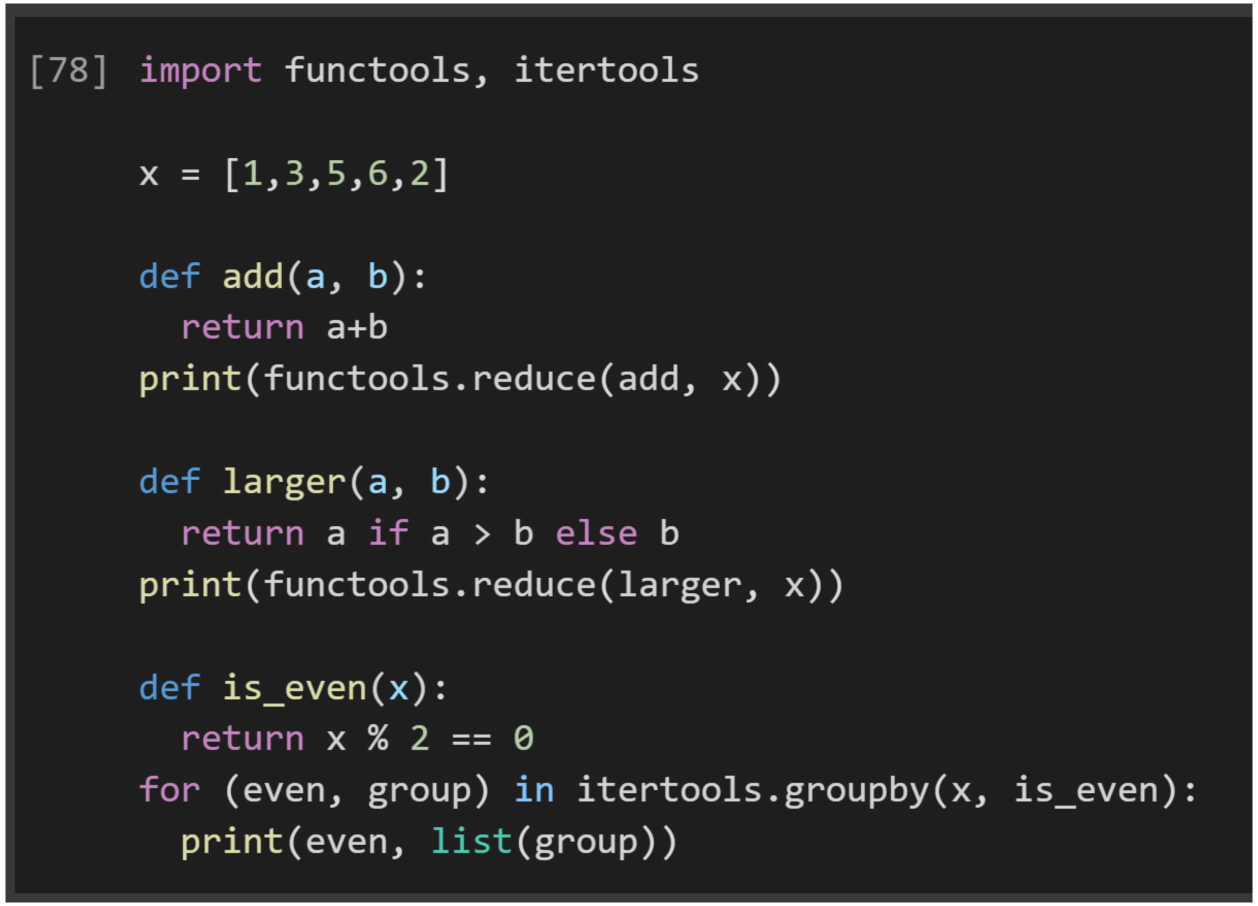
```
[69] import functools, itertools

     x = [1,3,5,6,2]

     print(sum(x))
     print(functools.reduce(lambda a, b: a+b, x))
     print(list(itertools.accumulate(x)))

     print(max(x))
     print(functools.reduce(lambda a, b: a if a > b else b, x))

     for (even, group) in itertools.groupby(x, lambda x: x % 2 == 0):
       print(even, list(group))
```



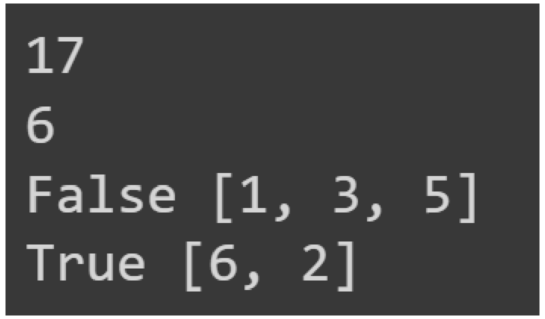
```
17
17
[1, 4, 9, 15, 17]
6
6
False [1, 3, 5]
True [6, 2]
```

Reduce, Groupby (non-lambda)

- If lambdas were overwhelming
- Reduce is a 2 argument function, groupby is 1 argument boolean function

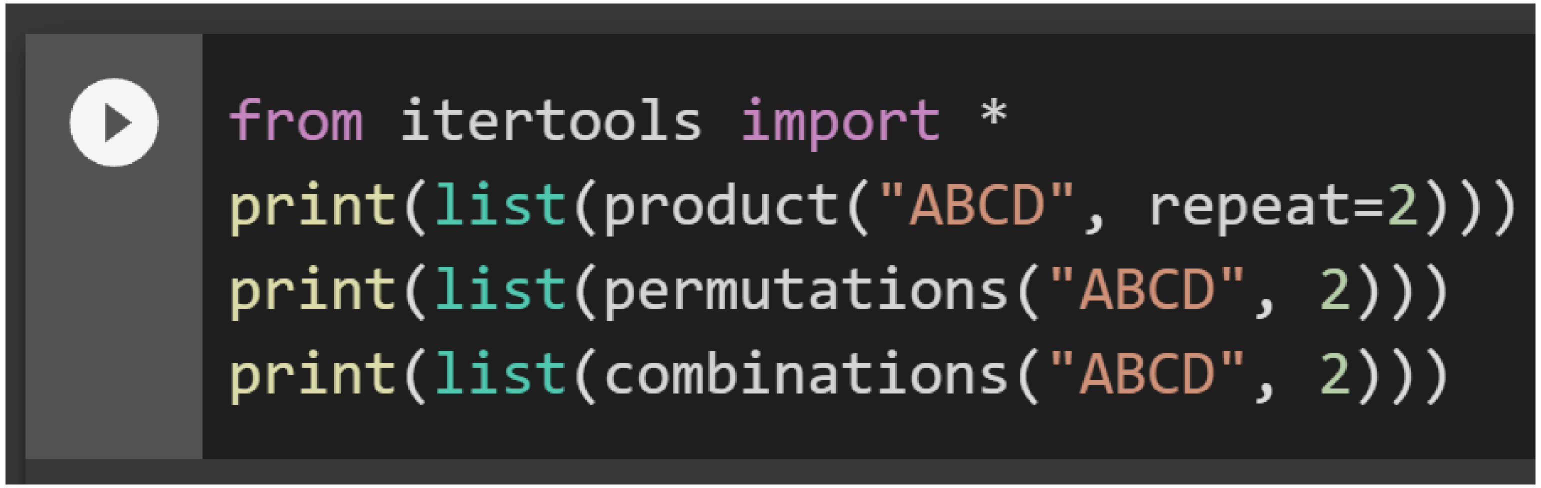


```
[78] import functools, itertools

     x = [1,3,5,6,2]

     def add(a, b):
       return a+b
     print(functools.reduce(add, x))

     def larger(a, b):
       return a if a > b else b
     print(functools.reduce(larger, x))

     def is_even(x):
       return x % 2 == 0
     for (even, group) in itertools.groupby(x, is_even):
       print(even, list(group))
```



```
17
6
False [1, 3, 5]
True [6, 2]
```

Product, Permutations, Combinations

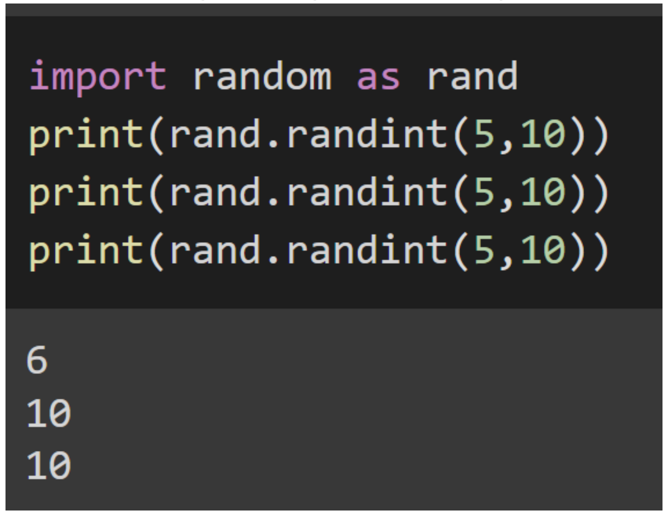


```python
from itertools import *
print(list(product("ABCD", repeat=2)))
print(list(permutations("ABCD", 2)))
print(list(combinations("ABCD", 2)))
```

```
[('A', 'A'), ('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'A'), ('B', 'B'), ('B', 'C'), ('B',
'D'), ('C', 'A'), ('C', 'B'), ('C', 'C'), ('C', 'D'), ('D', 'A'), ('D', 'B'), ('D', 'C'),
('D', 'D’)]

[('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'A'), ('B', 'C'), ('B', 'D'), ('C', 'A'), ('C',
'B'), ('C', 'D'), ('D', 'A'), ('D', 'B'), ('D', 'C’)]

[('A', 'B'), ('A', 'C'), ('A', 'D'), ('B', 'C'), ('B', 'D'), ('C', 'D’)]
```

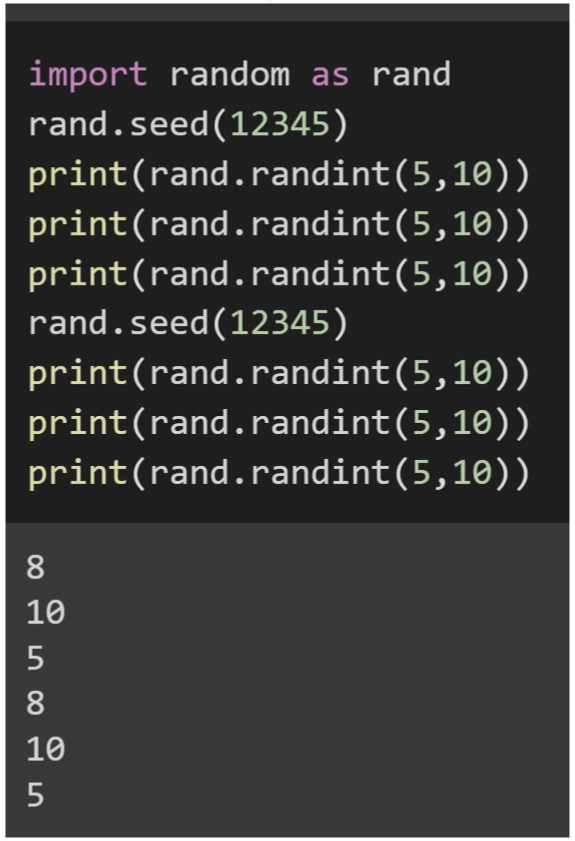
Random

- Psuedo-random number generation -> ‘pseudo’ because everything is a numerical sequence beginning at some **seed**
- **import random as rand**
- **rand.seed(<seed>)** -> is used to set this starting point if we want consistent behaviour each time program is run
- **randint** is single integer, **randrange** can allow you to select integer from dictate consistent range



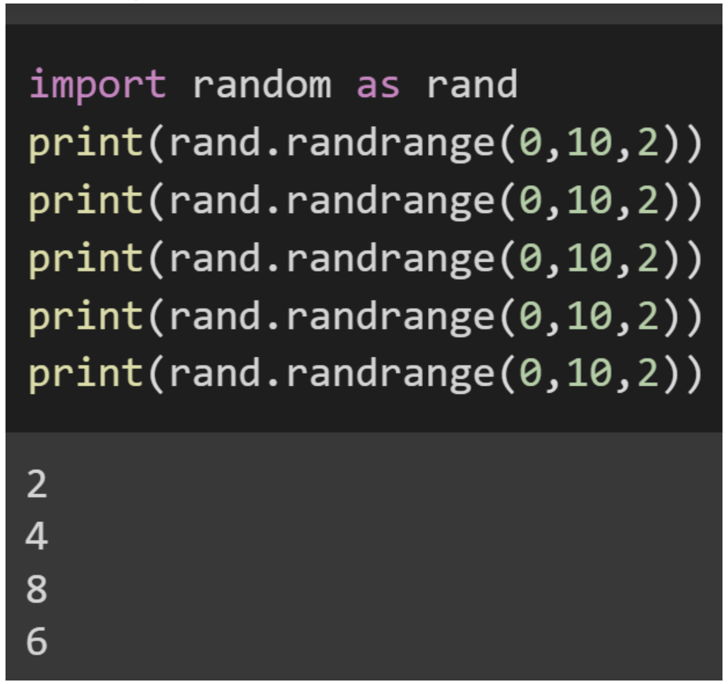
```python
import random as rand
print(rand.randint(5,10))
print(rand.randint(5,10))
print(rand.randint(5,10))
```

```
6
10
10
```



```python
import random as rand
rand.seed(12345)
print(rand.randint(5,10))
print(rand.randint(5,10))
print(rand.randint(5,10))
rand.seed(12345)
print(rand.randint(5,10))
print(rand.randint(5,10))
print(rand.randint(5,10))
```

```
8
10
5
8
10
5
```

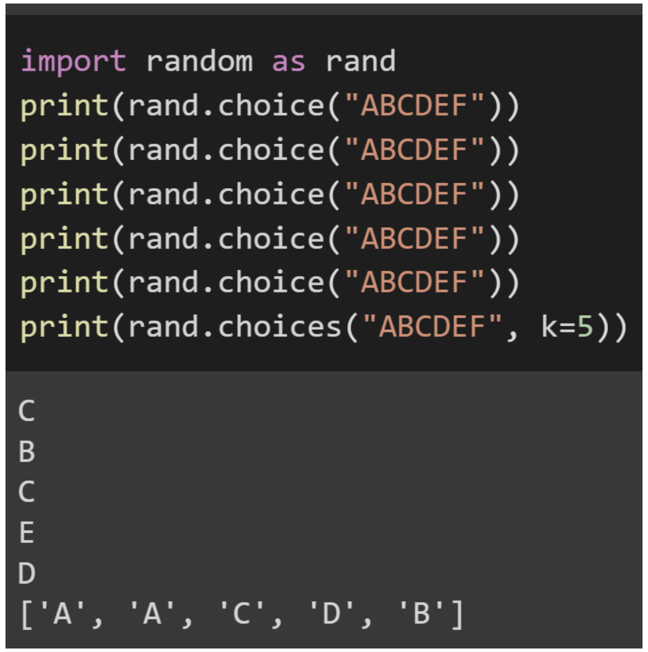


```python
import random as rand
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
print(rand.randrange(0,10,2))
```

```
2
4
8
6
```

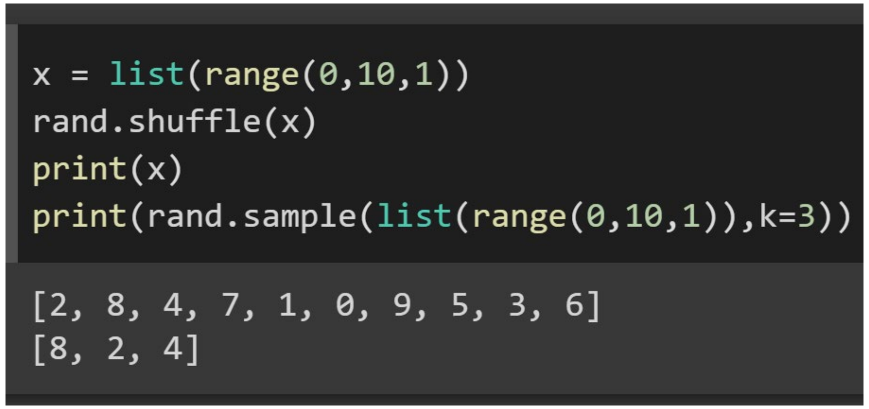
Random

- **rand.choice()** lets you select from a collection, and **choices**() a collection from a collection (valid to re-select things)
- **rand.shuffle()** will randomly permutate your collection, rand sample is like **choices**() without replacement (select each item once)
- **rand.random()** -> random real number between 0 and 1
- **rand.uniform()** -> random real number in dictate range
- **rand.normalvariate()** -> one of a number of distribution based random real number range selectors



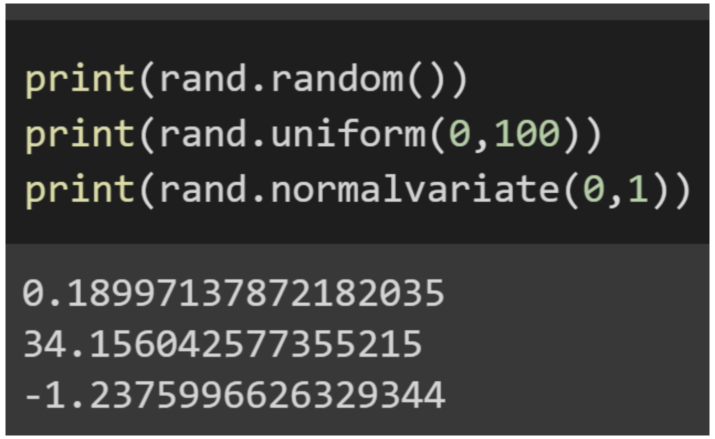
```python
import random as rand
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choice("ABCDEF"))
print(rand.choices("ABCDEF", k=5))
```

```
C
B
C
E
D
['A', 'A', 'C', 'D', 'B']
```



```python
x = list(range(0,10,1))
rand.shuffle(x)
print(x)
print(rand.sample(list(range(0,10,1)),k=3))
```

```
[2, 8, 4, 7, 1, 0, 9, 5, 3, 6]
[8, 2, 4]
```



```python
print(rand.random())
print(rand.uniform(0,100))
print(rand.normalvariate(0,1))
```

```
0.18997137872182035
34.156042577355215
-1.2375996626329344
```

Onward to … Libraries

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY