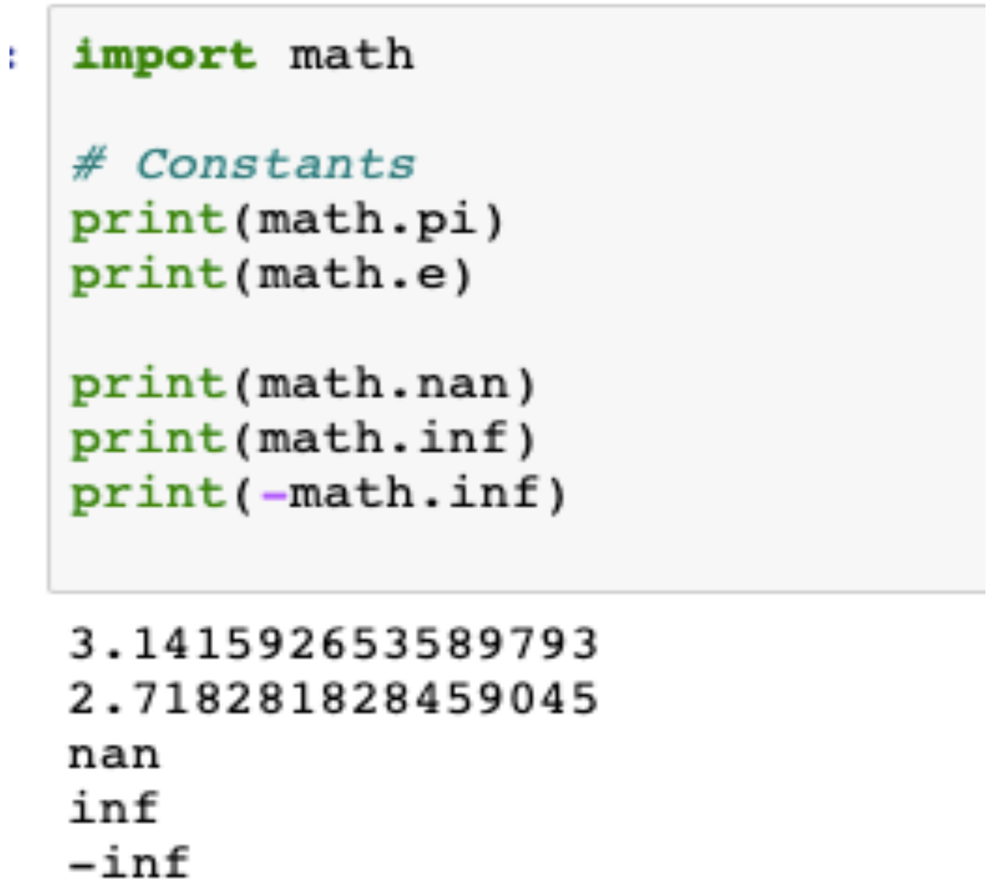
Python - Math

.

Ceiling, floor, and constants

- There are several built-in modules available in Python.
- In order to use them, we need to import them into our file by using import method.

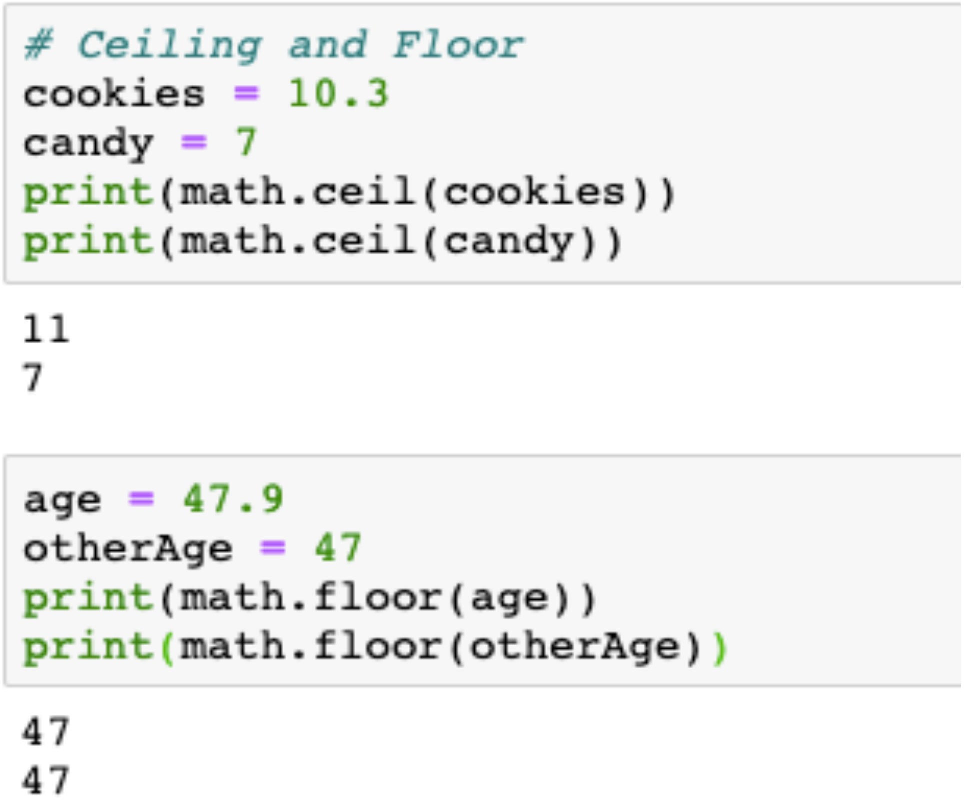


```
import math

# Constants
print(math.pi)
print(math.e)

print(math.nan)
print(math.inf)
print(-math.inf)
```

```
3.141592653589793
2.718281828459045
nan
inf
-inf
```

Ceiling, floor, and constants

- Here are some examples of ceiling and floor functions
- Cookies is equal to 10.3 here, but no one can actually have 10.3 cookies, Ceiling function here brings it up to 11 cookies.



```
# Ceiling and Floor
cookies = 10.3
candy = 7
print(math.ceil(cookies))
print(math.ceil(candy))
```

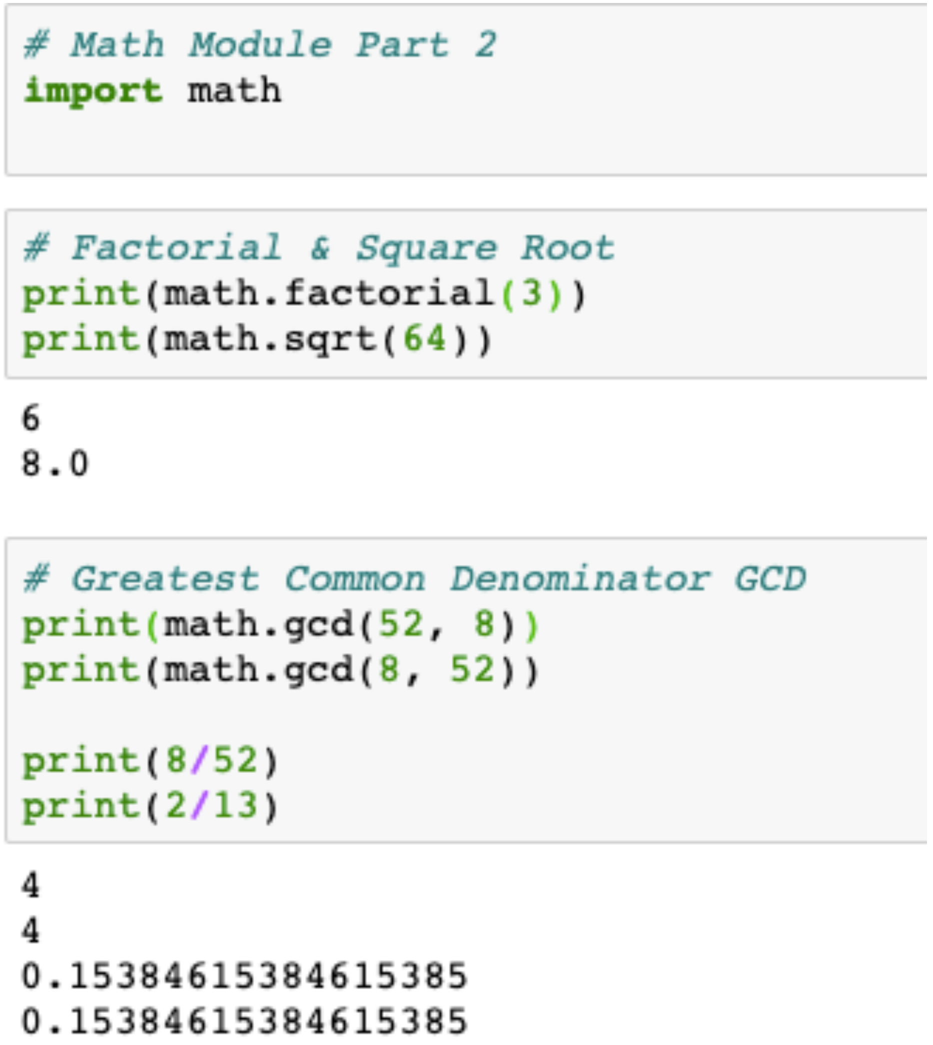
```
11
7
```

```
age = 47.9
otherAge = 47
print(math.floor(age))
print(math.floor(otherAge))
```

```
47
47
```

Factorial, square root, and GCD

- To use the factorial function, just use ‘math.factorial’ to get the factorial of any number
- ’math.sqrt’ for getting a square root of any number
- ‘math.gcd’ for getting the greatest common denominator between two numbers



```
# Math Module Part 2
import math
```

```
# Factorial & Square Root
print(math.factorial(3))
print(math.sqrt(64))
```

```
6
8.0
```

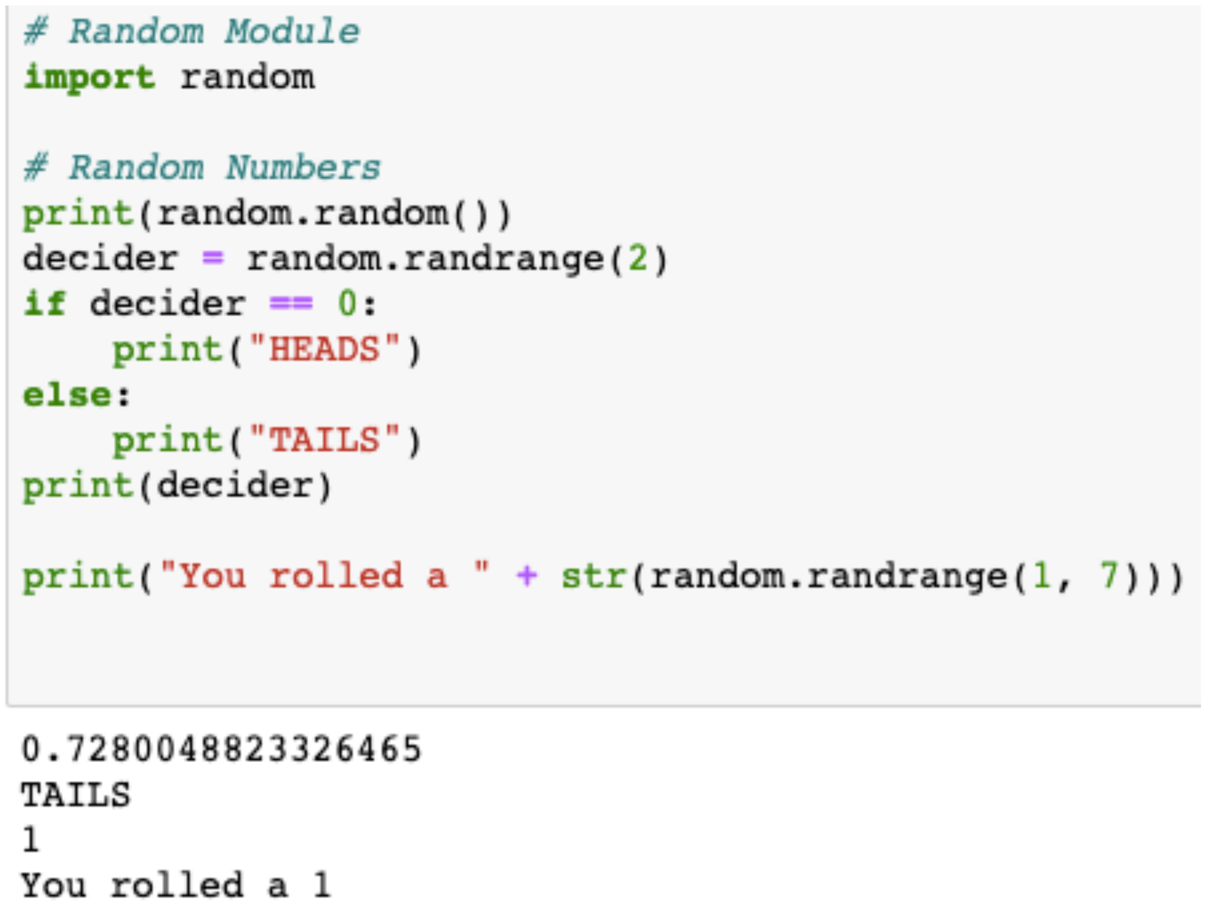
```
# Greatest Common Denominator GCD
print(math.gcd(52, 8))
print(math.gcd(8, 52))

print(8/52)
print(2/13)
```

```
4
4
0.15384615384615385
0.15384615384615385
```

Python random

- The random module allows you to create random values and generate random choices

- Printing random.random gives a random number from zero and one



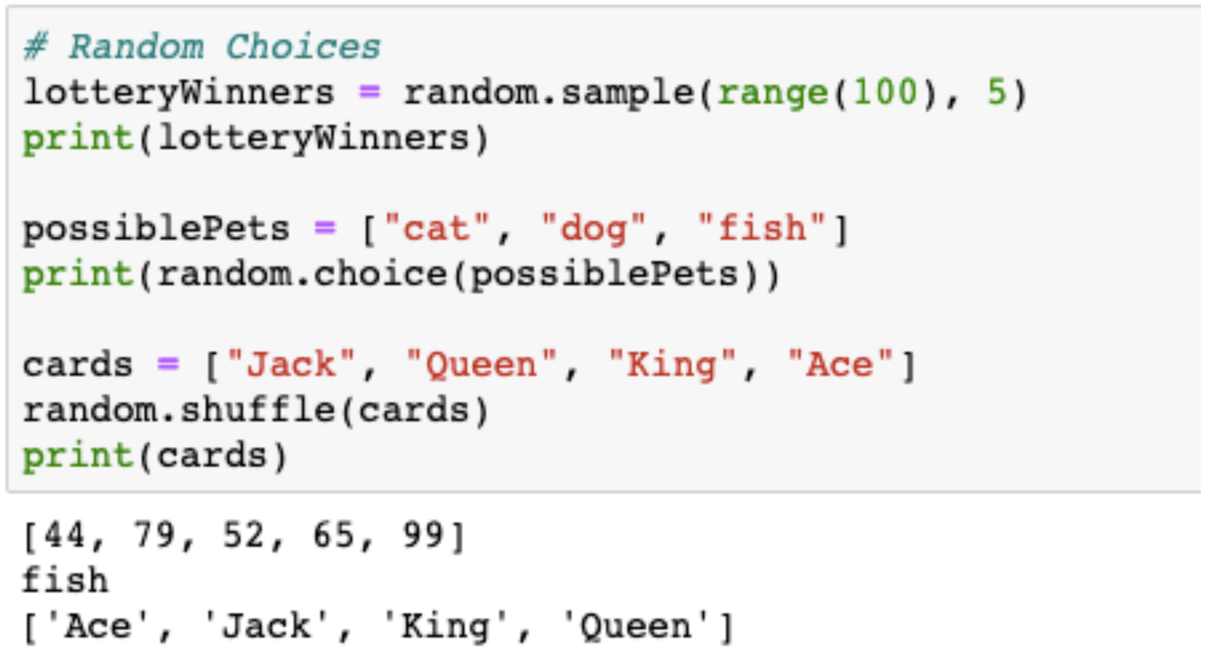
```python
# Random Module
import random

# Random Numbers
print(random.random())
decider = random.randrange(2)
if decider == 0:
    print("HEADS")
else:
    print("TAILS")
print(decider)

print("You rolled a " + str(random.randrange(1, 7)))
```

```
0.7280048823326465
TAILS
1
You rolled a 1
```

Python random

- Random module can also help us with random choice

- ‘random.sample’ , ‘random.choice’ and ‘random.shuffle’ are some of the functions which can be used in python

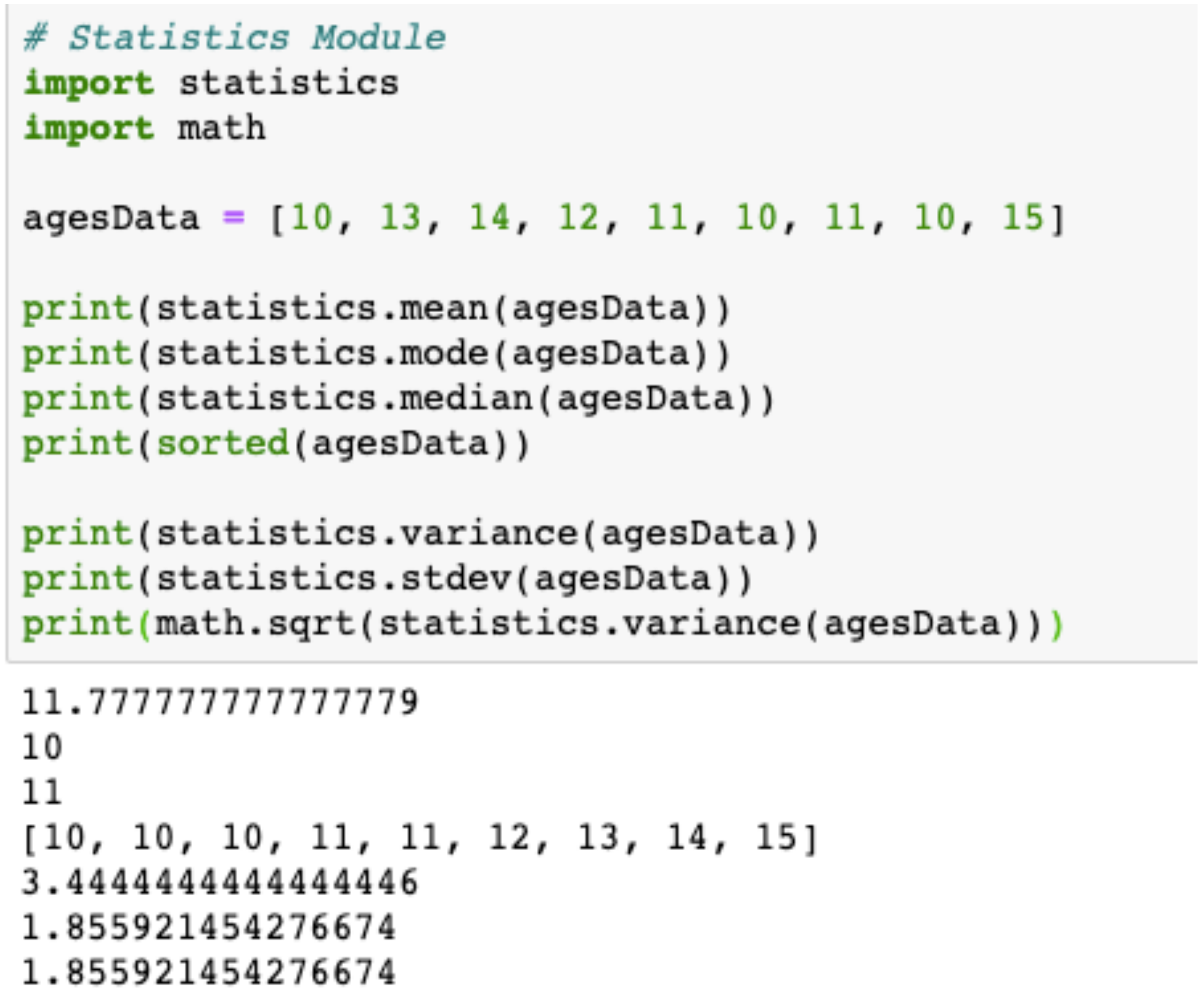


```python
# Random Choices
lotteryWinners = random.sample(range(100), 5)
print(lotteryWinners)

possiblePets = ["cat", "dog", "fish"]
print(random.choice(possiblePets))

cards = ["Jack", "Queen", "King", "Ace"]
random.shuffle(cards)
print(cards)
```

```
[44, 79, 52, 65, 99]
fish
['Ace', 'Jack', 'King', 'Queen']
```

Calculating statistics with Python

- The statistics module allows us to calculate common statistics like the mean, mode, standard deviation, etc including some more complicated things as well.
- Statistics have many built-in functions like mean, median, mode etc

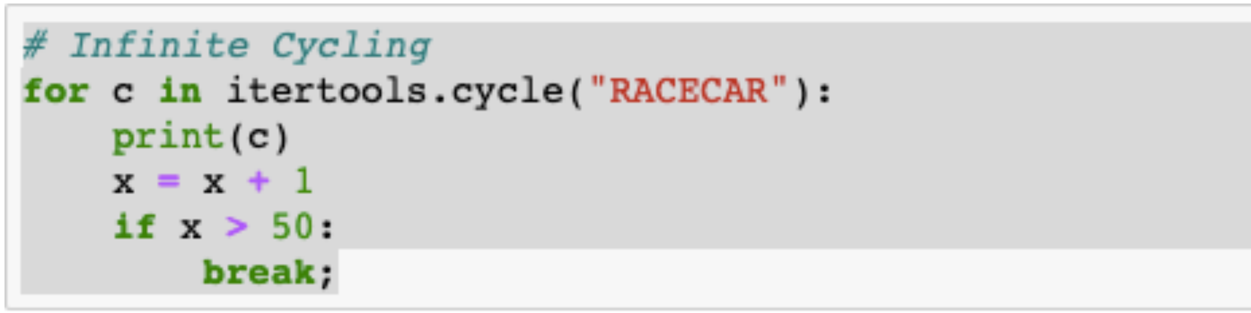


```
# Statistics Module
import statistics
import math

agesData = [10, 13, 14, 12, 11, 10, 11, 10, 15]

print(statistics.mean(agesData))
print(statistics.mode(agesData))
print(statistics.median(agesData))
print(sorted(agesData))

print(statistics.variance(agesData))
print(statistics.stdev(agesData))
print(math.sqrt(statistics.variance(agesData)))
```

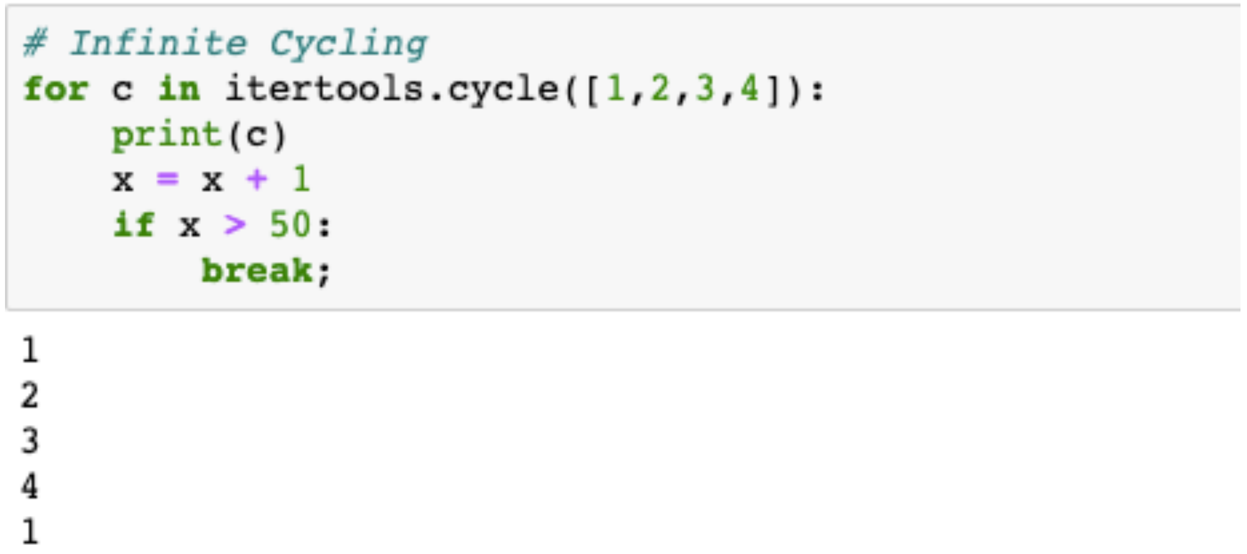
```
11.777777777777779
10
11
[10, 10, 10, 11, 11, 12, 13, 14, 15]
3.4444444444444446
1.855921454276674
1.855921454276674
```

Iterators with itertools: Infinite processes

- We can also infinitely cycle through an input using Itertools
- Cycle input also takes numbers as well
- Using the count, cycle, and repeat methods in the Itertools module allows you to loop through different data very efficiently while making your code shorter and more readable.



```python
# Infinite Cycling
for c in itertools.cycle("RACECAR"):
    print(c)
    x = x + 1
    if x > 50:
        break;
```

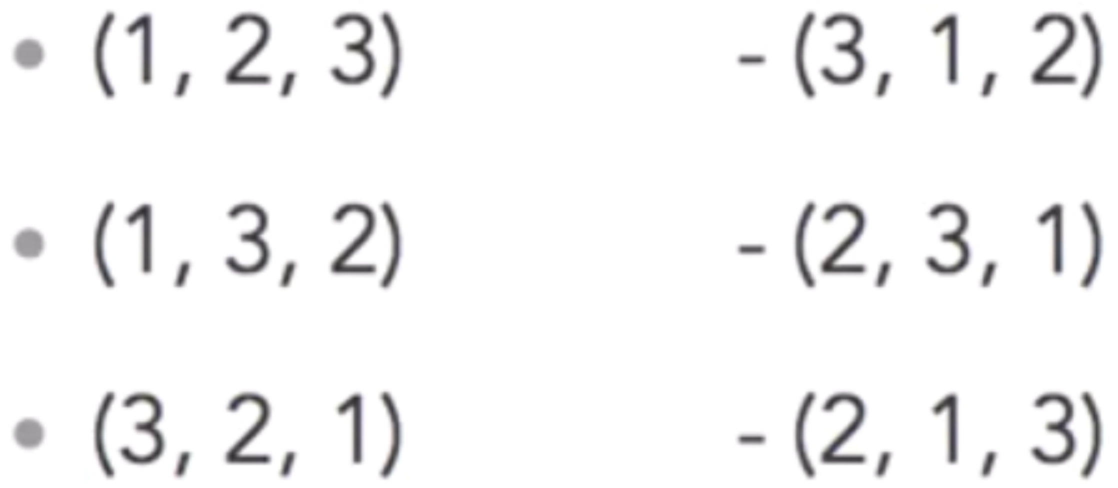


```python
# Infinite Cycling
for c in itertools.cycle([1,2,3,4]):
    print(c)
    x = x + 1
    if x > 50:
        break;
```

```
1
2
3
4
1
```

Permutations and combinations

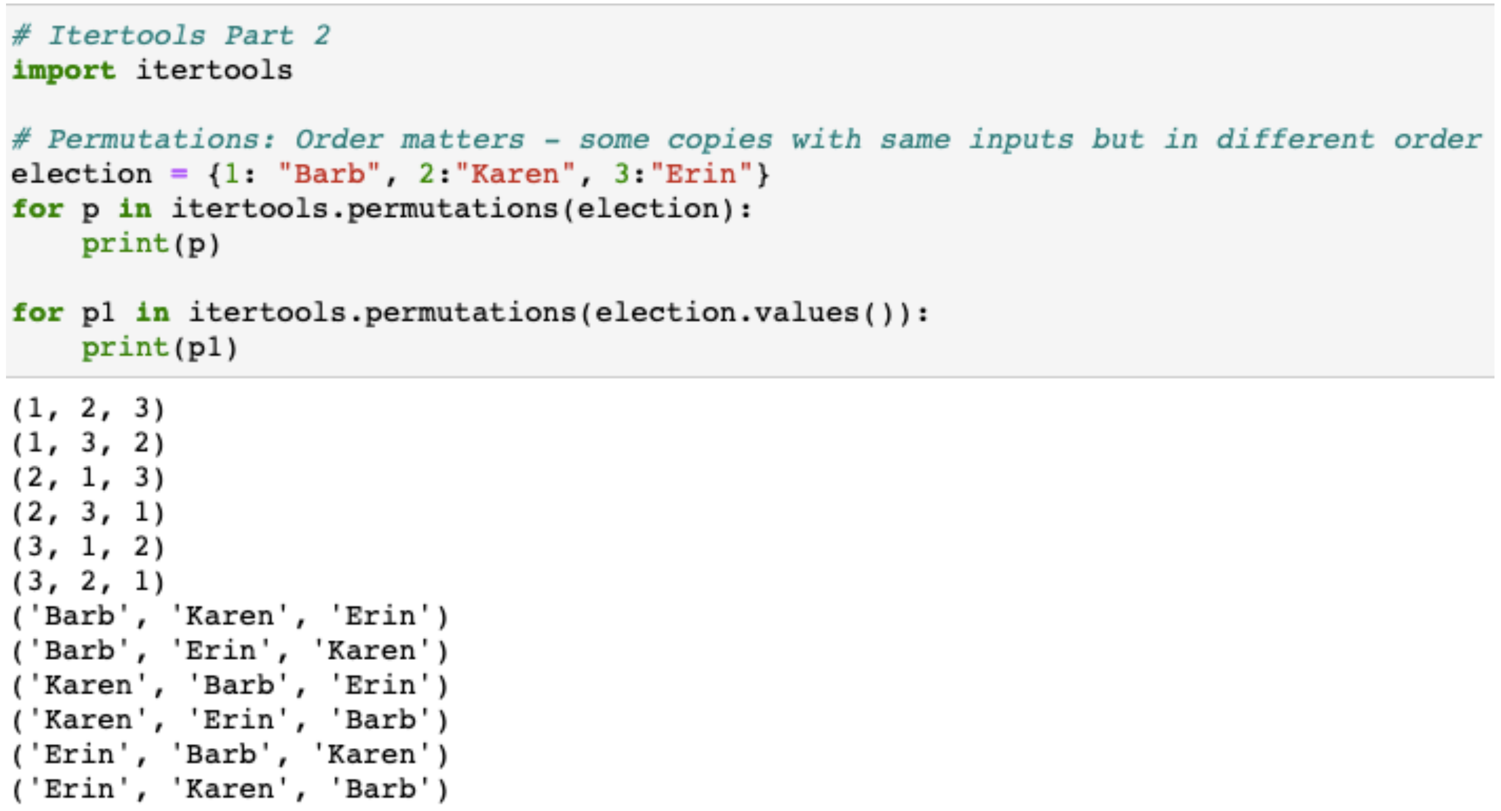
- A way, especially one of several possible variations in which a set or number of things can be ordered, or arranged
- Permutations of [1,2,3] - same items and different order



- (1, 2, 3) - (3, 1, 2)
- (1, 3, 2) - (2, 3, 1)
- (3, 2, 1) - (2, 1, 3)

Permutations and combinations

- ‘itertools.permutations’ gives all the possible orderings



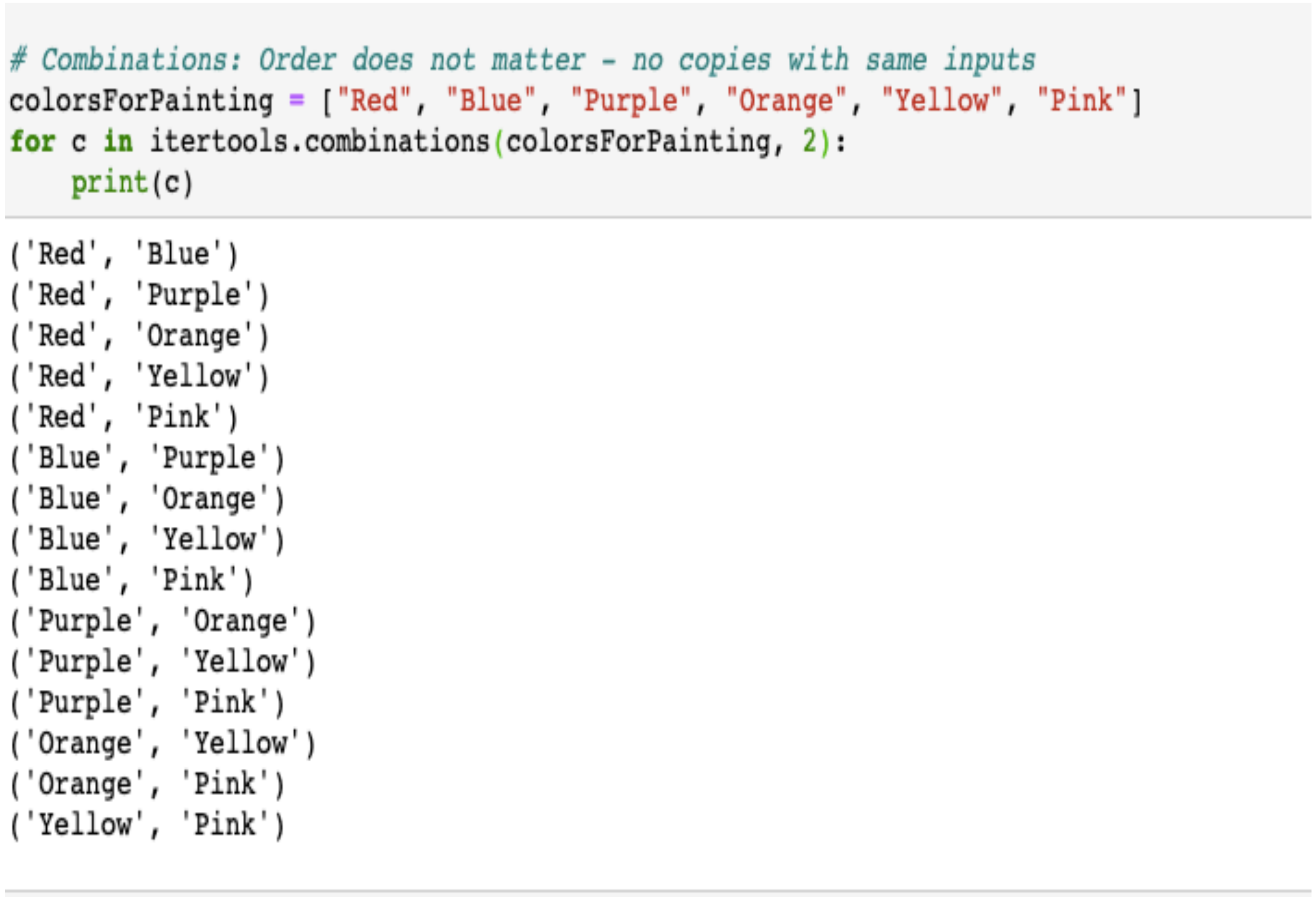
```
# Itertools Part 2
import itertools

# Permutations: Order matters - some copies with same inputs but in different order
election = {1: "Barb", 2:"Karen", 3:"Erin"}
for p in itertools.permutations(election):
    print(p)

for p1 in itertools.permutations(election.values()):
    print(p1)
```

```
(1, 2, 3)
(1, 3, 2)
(2, 1, 3)
(2, 3, 1)
(3, 1, 2)
(3, 2, 1)
('Barb', 'Karen', 'Erin')
('Barb', 'Erin', 'Karen')
('Karen', 'Barb', 'Erin')
('Karen', 'Erin', 'Barb')
('Erin', 'Barb', 'Karen')
('Erin', 'Karen', 'Barb')
```

Permutations and combinations

- In Combinations, no set has the exact same element as another

- ’itertools.combinations’ is used to get different combinations of the data



```
# Combinations: Order does not matter - no copies with same inputs
colorsForPainting = ["Red", "Blue", "Purple", "Orange", "Yellow", "Pink"]
for c in itertools.combinations(colorsForPainting, 2):
    print(c)
```

```
('Red', 'Blue')
('Red', 'Purple')
('Red', 'Orange')
('Red', 'Yellow')
('Red', 'Pink')
('Blue', 'Purple')
('Blue', 'Orange')
('Blue', 'Yellow')
('Blue', 'Pink')
('Purple', 'Orange')
('Purple', 'Yellow')
('Purple', 'Pink')
('Orange', 'Yellow')
('Orange', 'Pink')
('Yellow', 'Pink')
```