63_iterables

March 18, 2022

1 Iterables

```
[ ]: words = "Why sometimes I have believed as many as six impossible things before␣
      ↪breakfast".split()
     words
```

```
[ ]: ['Why',
      'sometimes',
      'I',
      'have',
      'believed',
      'as',
      'many',
      'as',
      'six',
      'impossible',
      'things',
      'before',
      'breakfast']
```

1.1 List Comprehensions

```
[ ]: [len(word) for word in words]
```

```
[ ]: [3, 9, 1, 4, 8, 2, 4, 2, 3, 10, 6, 6, 9]
```

```
[ ]: from math import factorial

     f = [len(str(factorial(x))) for x in range(20)]
     f
```

```
[ ]: [1, 1, 1, 1, 2, 3, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 18]
```

```
[ ]: print(type(f))
```

```
<class 'list'>
```

1.2 Set Comprehensions

```
[ ]: # Applying comprehension in sets
     s = {len(str(factorial(x))) for x in range(20)}
     s
```

```
[ ]: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16, 18}
```

1.3 Dictionary(dict) Comprehensions

```
[ ]: # Dictionaries of cities
     country_to_capital = {
         'United Kingdom': 'London',
         'Brazil': 'Brasilia',
         'Morocco': 'Rabat',
         'Sweden': 'Stockholm',
         'Pakistan': 'Islamabad'
     }
```

```
[ ]: # for country, capital in country_to_capital.items()
     country_to_capital.items()
     countries = []
     capitals = []
     for country, capital in country_to_capital.items():
         countries.append(country)
         capitals.append(capital)
     print(countries)
     print(capitals)
     # print (country)
     # print (capital)
```

```
['United Kingdom', 'Brazil', 'Morocco', 'Sweden', 'Pakistan']
['London', 'Brasilia', 'Rabat', 'Stockholm', 'Islamabad']
```

```
[ ]: # Syntax
     # {
     #     key_expression(item): value_expression(item)
     #     for item in iterable
     # }

     # Applying comprehension on dictionary
     # capital_to_country = {
     #     capital: country
     #     for country, capital in country_to_capital.items()
     # }

     # capital_to_country
```

```
[ ]: capital_to_country = {
         capital: country
         for country, capital in country_to_capital.items()
     }

     capital_to_country
```

```
[ ]: {'London': 'United Kingdom',
      'Brasilia': 'Brazil',
      'Rabat': 'Morocco',
      'Stockholm': 'Sweden',
      'Islamabad': 'Pakistan'}
```

```
[ ]: words = ['hi', "hello", "foxtrot", "hotel"]
     # Applying comprehension from list to dict
     # providing same values of key will update the existing value (so h has been␣
      ↪overwritten 2 times and last hotel's h is preserved
     {
         x[0]: x
         for x in words
     }
```

```
[ ]: {'h': 'hotel', 'f': 'foxtrot'}
```

1.4 Filtering comprehensions

```
[ ]: from math import sqrt


     def is_prime(x):
         if x < 2:
             return False
         for i in range(2, int(sqrt(x)) + 1):
             if x % i == 0:
                 return False
             return True


     [x for x in range(101) if is_prime(x)]
```

```
[ ]: [5,
      7,
      9,
      11,
      13,
      15,
      17,
 19,
 21,
 23,
 25,
 27,
 29,
 31,
 33,
 35,
 37,
 39,
 41,
 43,
 45,
 47,
 49,
 51,
 53,
 55,
 57,
 59,
 61,
 63,
 65,
 67,
 69,
 71,
 73,
 75,
 77,
 79,
 81,
 83,
 85,
 87,
 89,
 91,
 93,
 95,
 97,
 99]
```

[ ]:
```
prime_square_divisors = {
    x * x: (1, x, x * x)
    for x in range(20) if is_prime(x)
}
prime_square_divisors
```

```
[ ]: {25: (1, 5, 25),
      49: (1, 7, 49),
      81: (1, 9, 81),
      121: (1, 11, 121),
      169: (1, 13, 169),
      225: (1, 15, 225),
      289: (1, 17, 289),
      361: (1, 19, 361)}
```

2 Iteration Protocols

```
[ ]: iterable = ['Spring', 'Summer', "Autumn", 'Winter']
     iterator = iter(iterable)
```

```
[ ]: # Everytime next is called on an iterator, it should return next value in the␣
      ↪iterator
     next(iterator)
```

```
[ ]: 'Spring'
```

```
[ ]: def first(an_iterable):
         an_iterator = iter(an_iterable)
         try:
             return next(an_iterator)
         except StopIteration:
             raise ValueError("itearble is empty")
```

```
[ ]: first(["1st", "2nd", "3rd"])
     # first({"1st", "2nd", "3rd", "4th"})
```

```
[ ]: '1st'
```

```
[ ]: # first({"1st", "2nd", "3rd"})
     first([1, 2, 3, 4])
```

```
[ ]: 1
```

```
[ ]: first(set())
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
e:
 ↪\learning\python\pluralsight\core-python-getting-started\iterables\63_iterables.
 ↪ipynb Cell 21' in first(an_iterable)
      <a href='vscode-notebook-cell:/e%3A/learning/python/pluralsight/
 ↪core-python-getting-started/iterables/63_iterables.ipynb#ch0000020?line=2'>3<
 ↪a> try:
```

```
----> <a href='vscode-notebook-cell:/e%3A/learning/python/pluralsight/
  ↪core-python-getting-started/iterables/63_iterables.ipynb#ch0000020?line=3'>4<
  ↪a>     return next(an_iterator)
      <a href='vscode-notebook-cell:/e%3A/learning/python/pluralsight/
  ↪core-python-getting-started/iterables/63_iterables.ipynb#ch0000020?line=4'>5<
  ↪a> except StopIteration:

StopIteration:

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call last)
e:
  ↪\learning\python\pluralsight\core-python-getting-started\iterables\63_iterables.
  ↪ipynb Cell 24' in <cell line: 1>()
----> <a href='vscode-notebook-cell:/e%3A/learning/python/pluralsight/
  ↪core-python-getting-started/iterables/63_iterables.ipynb#ch0000023?line=0'>1<
  ↪a> first(set())


e:
  ↪\learning\python\pluralsight\core-python-getting-started\iterables\63_iterables.
  ↪ipynb Cell 21' in first(an_iterable)
      <a href='vscode-notebook-cell:/e%3A/learning/python/pluralsight/
  ↪core-python-getting-started/iterables/63_iterables.ipynb#ch0000020?line=3'>4<
  ↪a>     return next(an_iterator)
      <a href='vscode-notebook-cell:/e%3A/learning/python/pluralsight/
  ↪core-python-getting-started/iterables/63_iterables.ipynb#ch0000020?line=4'>5<
  ↪a> except StopIteration:
----> <a href='vscode-notebook-cell:/e%3A/learning/python/pluralsight/
  ↪core-python-getting-started/iterables/63_iterables.ipynb#ch0000020?line=5'>6<
  ↪a>     raise ValueError("itearble is empty")

ValueError: itearble is empty
```

2.1 Generator Expression

[ ]:
```
# Generating sum of first million squares by using generator - A lot less␣
 ↪memory consumption
sum(x * x for x in range(1, 1000001))
```

[ ]:
```
333333833333500000
```

[ ]:
```
name = 'Abid Ali'
name.title()
```

[ ]:
```
'Abid Ali'
```

2.2 Any and All

```
[ ]: # Any and All
     any([False, False, True])
```

```
[ ]: all([False, False, True])
```

```
[ ]: False
```

```
[ ]: all(name == name.title() for name in ['London', 'Paris', 'Tokyo', 'New York'])
```

```
[ ]: True
```

2.3 zip

Synchronize iteration across two more more iterables

```
[ ]: temperature_sunday = [12, 14, 15, 15, 17, 21, 22, 22, 23, 22, 20, 18]
     temperature_monday = [13, 14, 14, 14, 16, 20, 21, 22, 22, 21, 19, 17]
     temperature_tuesday = [2, 2, 3, 7, 9, 10, 11, 12, 10, 9, 8, 8]

     # returns tuple
     for item in zip(temperature_sunday, temperature_monday):
         print(item)

     # We can do tuple unpacking in for loop
     for sun, mon in zip(temperature_sunday, temperature_monday):
         print(f"average = {(sun + mon) / 2}")

     for temps in zip(temperature_sunday, temperature_monday, temperature_tuesday):
         print(f"min = {min(temps):4.1f}, max={max(temps):4.1f}, average={sum(temps)␣
      ↪/ len(temps):4.1f}")
```

```
(12, 13)
(14, 14)
(15, 14)
(15, 14)
(17, 16)
(21, 20)
(22, 21)
(22, 22)
(23, 22)
(22, 21)
(20, 19)
(18, 17)
average = 12.5
average = 14.0
average = 14.5
average = 14.5
average = 16.5
average = 20.5
average = 21.5
average = 22.0
average = 22.5
average = 21.5
average = 19.5
average = 17.5
min = 2.0, max=13.0, average= 9.0
min = 2.0, max=14.0, average=10.0
min = 3.0, max=15.0, average=10.7
min = 7.0, max=15.0, average=12.0
min = 9.0, max=17.0, average=14.0
min = 10.0, max=21.0, average=17.0
min = 11.0, max=22.0, average=18.0
min = 12.0, max=22.0, average=18.7
min = 10.0, max=23.0, average=18.3
min = 9.0, max=22.0, average=17.3
min = 8.0, max=20.0, average=15.7
min = 8.0, max=18.0, average=14.3
```

[ ]:
```python
for temps in zip(temperature_sunday, temperature_monday, temperature_tuesday):
    print(f"min = {min(temps):4.1f}, max={max(temps):4.1f}, average={sum(temps)␣
 ↪/ len(temps):4.1f}")
```

```
min =  2.0, max=13.0, average= 9.0
min =  2.0, max=14.0, average=10.0
min =  3.0, max=15.0, average=10.7
min =  7.0, max=15.0, average=12.0
min =  9.0, max=17.0, average=14.0
min = 10.0, max=21.0, average=17.0
min = 11.0, max=22.0, average=18.0
min = 12.0, max=22.0, average=18.7
min = 10.0, max=23.0, average=18.3
min =  9.0, max=22.0, average=17.3
min =  8.0, max=20.0, average=15.7
min =  8.0, max=18.0, average=14.3
```

[ ]:
```python
from itertools import chain

temperatures = chain(temperature_sunday, temperature_monday,␣
 ↪temperature_tuesday)
# check if all temperatures are above freezing point?
all(t > 0 for t in temperatures)
```

[ ]:
```
True
```