3.Complex_data_types

November 29, 2018

1 Complex data structures

1. Types with 1 value
    - int
    - float
    - str
    - bool
2. Types with several values (collections)
    - list
    - tuple
    - set
    - dict

1.1 List

List - is a collection of elements

```
In [291]: polymerases = ['RdRP', 'DdRP', 'RdDP']
          polymerases

Out[291]: ['RdRP', 'DdRP', 'RdDP']
```

1.2 Characteristics of list

- sequence - items in list are go in sequential ordered
- mutability - you can change list (add items, remove them, change order)

1.3 list creation

- `[]` - part of language syntax
- `list()` - list constructor (later about it)

```
In [292]: a = []
          a

Out[292]: []
```

```
In [293]: b = list()
          b

Out[293]: []

In [294]: # Brackets usually used to create lists from a set of primitive values
          my_numbers = [1, 2, 3]
          my_numbers

Out[294]: [1, 2, 3]

In [295]: # Constructor usually used to create lists from other collections (later about it)
          my_numbers = list((-3, 2, 10.5))
          my_numbers

Out[295]: [-3, 2, 10.5]
```

1.4 Getting elements from list (indexing)

```
In [296]: # Create our list
          numbers = [1, 2, 3, 4, 5]
          numbers

Out[296]: [1, 2, 3, 4, 5]

In [297]: # Take 1 element from a list
          numbers[0], numbers[1], numbers[2]

Out[297]: (1, 2, 3)

In [298]: numbers[-1], numbers[-2], numbers[-3], numbers[-0]

Out[298]: (5, 4, 3, 1)

In [299]: # Take slices of list
          numbers[0:2]

Out[299]: [1, 2]

In [300]: numbers[1:4]

Out[300]: [2, 3, 4]
```

1.5 Morphology of Slicing

`list[start:stop:step]` * elements are taken including from start up to stop but excluding element with index stop - half closed interval in math $[start; stop)$ * start is 0 by default - `list[0:2] == list[:2]` * stop is equal to $length_of_list - 1$ by default - `list[3:length_of_list - 1] == list[3:]` * both index can be omitted - `list[::]` * step is equal to 1 by defaut, i.e. subsequent values are taken - `list[::1] == list[::]`

```
In [301]: # Let's practice
          numbers

Out[301]: [1, 2, 3, 4, 5]

In [302]: numbers[3:]

Out[302]: [4, 5]

In [303]: numbers[:-2]

Out[303]: [1, 2, 3]

In [304]: numbers[::2]

Out[304]: [1, 3, 5]

In [305]: numbers[::-1]

Out[305]: [5, 4, 3, 2, 1]
```

1.6 Functions applicable to lists

Just a few of them

- `len()` - awesome function which returns number of elements in collection
- `sorted()` - function to sort collection in some order (natural ascending by default)

```
In [306]: len(numbers)

Out[306]: 5

In [307]: numbers[len(numbers) // 2]

Out[307]: 3

In [308]: numbers = [1, 2, 5, 3, 4]
          sorted(numbers)

Out[308]: [1, 2, 3, 4, 5]
```

1.7 List methods

First of all methods are like a functions, but associated with object. They have slightly different notation

```
list.method()
```

- `append(element)` - append element to the end of list
- `remove(element)` - remove element from the list
- `extend(list2)` - add list2 content to the end of list - analogous to + with lists

- `index(element)` - get index of element 1st occurence in a list
- `count(element)` - count occurences of element in a list

```
In [309]: numbers

Out[309]: [1, 2, 5, 3, 4]

In [310]: numbers.append(10)
          numbers

Out[310]: [1, 2, 5, 3, 4, 10]

In [311]: numbers.remove(2)
          numbers

Out[311]: [1, 5, 3, 4, 10]

In [312]: numbers + [1, 2, 3]

Out[312]: [1, 5, 3, 4, 10, 1, 2, 3]

In [313]: numbers

Out[313]: [1, 5, 3, 4, 10]

In [314]: print(numbers.extend([1, 2, 3]))

None


In [315]: numbers

Out[315]: [1, 5, 3, 4, 10, 1, 2, 3]

In [316]: numbers.index(1)

Out[316]: 0

In [317]: numbers.index(12)

        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-317-81330119c69e> in <module>()
    ----> 1 numbers.index(12)


        ValueError: 12 is not in list

In [318]: numbers.count(3)

Out[318]: 2

In [319]: numbers.count(45)

Out[319]: 0
```

1.8 List mutability

You have already seen it with methods like `pop` and `remove`

```
In [325]: # Also you can do such things
          print(numbers)
          numbers[1] = 15
          print(numbers)

[5, 3, 4, 10, 1, 2]
[5, 15, 4, 10, 1, 2]
```

1.9 Quite a deeper look at language structure

Some covered types are "constant" in a sense that their values are immutable and when you value of variable, new value is assigned to a variable with no interference with other variables. It's not the case with lists.

```
In [326]: a = 3
          b = a
          print('a is', a)
          print('b is', b)

a is 3
b is 3


In [327]: a = 10
          print('a is', a, 'now')
          print('b is still', b)

a is 10 now
b is still 3


In [328]: a = [1, 2, 3]
          b = a
          print('a is', a)
          print('b is', b)

a is [1, 2, 3]
b is [1, 2, 3]


In [329]: a.append(100)
          print('a is', a, 'now')
          print('b has changed too - ', b)
```

```
a is [1, 2, 3, 100] now
b has changed too -  [1, 2, 3, 100]
```

You refer to different objects when you use 1 of the follow things

```
In [330]: b = a[:]
          b = a.copy()
          b = list(a)
```

1.10 Tuples

Similar to lists in many aspects but immutable

Creation of tuple: * from several elements - (value1, value2, value3) * from other collection - tuple([value1, value2, value3]) * tuple with 1 element - (value1,) - due to ambiguity of (value1)

Benefits of immutability * really awesome for parallelization (no data corruption) * implies less memory overhead to store values * other guys can't change your data

```
In [331]: cool_tuple = (1, 2, 3)
          awesome_list = list(x)

In [332]: print("Memory for list is", sys.getsizeof(awesome_list))
          print("Memory for tuple is", sys.getsizeof(cool_tuple))

Memory for list is 112
Memory for tuple is 72
```

1.11 Tuple methods

Everything is similar to list for these methods * `index(element)` - get index of element in a tuple * `count(element)` - get number of occurences of element in a tuple

```
In [333]: cool_tuple.index(3)

Out[333]: 2

In [334]: cool_tuple.count(2)

Out[334]: 1
```

1.12 Sets

Interesting collection which is * mutable - can be changed * unordered - elements in it can alternate their "indices" (it has no indices indeed) * holds only unique elems

It is a mathematical set

```
In [335]: # Ways to create
          # Empty set can be created only with set()
          imba_set = set()
          imba_set
```

```
Out[335]: set()

In [336]: # Set from elements
          elements_set = {1, 3, 2, 2, 3, 5}
          elements_set

Out[336]: {1, 2, 3, 5}

In [337]: # Set from other collection
          another_set = set([1, 2, 3])
          another_set

Out[337]: {1, 2, 3}

In [338]: set((1, 2, 2, 3, 2, 5))

Out[338]: {1, 2, 3, 5}

In [339]: set('ATGTGTGTAGAATATGT')

Out[339]: {'A', 'G', 'T'}

In [340]: print(list('ATGTGTGTAGAATATGT'))

['A', 'T', 'G', 'T', 'G', 'T', 'G', 'T', 'A', 'G', 'A', 'A', 'T', 'A', 'T', 'G', 'T']
```

1.13 Set methods

- `add(elem)` - add elem to a set if it is not in in already
- `remove(elem)` - remove elem from a set
- `update(colection2)` - set operation - make original set a union of set and colection2
- `intersect_update(colection2)` - set operation - make original set an intersection of set and colection2
- `difference_update(colection2)` - set operation - make original set a set without elements from colection2

```
In [341]: another_set.add(10)
          another_set

Out[341]: {1, 2, 3, 10}

In [342]: another_set.remove(2)
          another_set

Out[342]: {1, 3, 10}

In [343]: another_set.remove(2)
```

```
        ---------------------------------------------------------------------------

        KeyError                                  Traceback (most recent call last)

        <ipython-input-343-19ebac6688e6> in <module>()
    ----> 1 another_set.remove(2)


        KeyError: 2
```

```
In [344]: imba_set.update([1, 2])
          imba_set

Out[344]: {1, 2}

In [345]: imba_set.intersection_update((2, 3))
          imba_set

Out[345]: {2}

In [346]: imba_set.difference_update({2, 3, 5})
          imba_set

Out[346]: set()
```