Python 4: In-Built Functions & SOLID

IN608: Intermediate Application Development Concepts

Kaiako: Tom Clark & Grayson Orr

Last Session's Content

- More abstract data types
  - List as a stack
  - Stack class
  - List as a queue
  - Queue class
  - Circular queue

Today’s Content

- In-built functions
  - Enumerate
  - Reversed
  - Slice
  - Sorted
  - Zip
- SOLID
  - Single-responsibility principle
  - Open-closed principle
  - Liskov substitution principle
  - Interface segregation principle
  - Dependency inversion principle

In-Built Functions

Enumerate

- `enumerate(iterable, start=0)`
- Returns an enumerate object
- *iterable* must be an object which supports iteration
- Returns a tuple containing a count (default = 0) & values from iterating over *iterable*
- Resource: https://docs.python.org/3/library/functions.html#enumerate

```
seasons = ['Summer', 'Autumn', 'Winter', 'Spring']
seasons_enumerate = enumerate(seasons, start=1)
print(type(seasons_enumerate)) # <class enumerate>
print(list(seasons_enumerate)) # [(1, 'Summer'), (2, 'Autumn'), (3, 'Winter'), (4, 'Spring')]
```

Reversed

- `reversed(seq)`
- Returns a reverse iterator
- *seq* must be an object which has a `__reversed__()` method or supports the sequence protocol
- Resource: https://docs.python.org/3/library/functions.html#reversed

```
seasons = ['Summer', 'Autumn', 'Winter', 'Spring']
seasons_reversed = reversed(seasons)
pprint(type(seasons_reversed)) # <class list_reverseiterator>
print(list(seasons_reversed)) # ['Spring', 'Winter', 'Autumn', 'Summer']
```

Slice

- `slice(start, stop, step)`
- Returns a slice object representing the set of indices specified by `range(start, stop, step)`
- `start` & `step` arguments default to `None`
- Resource: https://docs.python.org/3/library/functions.html#slice

```
nums = [x for x in range(1, 11)]
start = slice(1)
start_stop = slice(0, 5)
start_stop_step = slice(0, 10, 2)
neg_step = slice(None, None, -1)
print(type(start)) # <class 'slice'>
print(nums[start]) # [1]
print(nums[start_stop]) # [1, 2, 3, 4, 5]
print(nums[start_stop_step]) # [1, 3, 5, 7, 9]
print(nums[neg_step]) # [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
print('Hello World!'[neg_step]) # !dlroW olleH
```

Sorted

- `sorted(iterable, key=None, reverse=False)`
- Returns a new sorted lists from elements in *iterable*
- Two optional arguments which must be specified as keyword arguments/kwargs
- Resource: https://docs.python.org/3/library/functions.html#sorted

```
def occurrence(item):
    return item.count('m')

seasons = ['Summer', 'Autumn', 'Winter', 'Spring']
seasons_sorted = sorted(seasons)
seasons_sorted_key = sorted(seasons, key=occurrence)
seasons_sorted_desc = sorted(seasons, reverse=True)
print(type(seasons_sorted)) # <class 'list'>
print(seasons_sorted) # ['Autumn', 'Spring', 'Summer', 'Winter']
print(seasons_sorted_key) # ['Winter', 'Spring', 'Autumn', 'Summer']
print(seasons_sorted_desc) # ['Winter', 'Summer', 'Spring', 'Autumn']
```

Zip

- `zip(*iterable)`
- Returns an iterator of tuples, where the nth tuple contains the nth element from each iterable
- The iterator stops when the smallest iterable in length is exhausted
- Resource: https://docs.python.org/3/library/functions.html#zip

```
first_names = ['Ari', 'Yunus', 'Ali']
last_names = ['Molina', 'Rennie', 'Rocha']
occupations = ['Professor', 'Associate Professor', 'Assistant Professor']
people = zip(first_names, last_names, occupations)
print(type(people)) # <class 'zip'>
print(list(people)) # [('Ari', 'Molina', 'Professor'), ('Yunus', 'Rennie', 'Associate Professor'), ('Ali', 'Rocha', 'Assistant Professor')]

months = ['Jan', 'Feb', 'Mar', 'Apr']
gross_per_month = [44611.00, 47976.00, 47535.00, 45383.00]
cost_per_month = [46893.00, 43157.00, 41164.00, 40761.00]
calculations = zip(months, gross_per_month, cost_per_month)
for m, g, c in calculations:
    net_profit = g - c
    print(f'Profit for {m}: {net_profit}') # Profit for Jan: -2282.0
                                           # Profit for Feb: 4819.0
                                           # Profit for Mar: 6371.0
                                           # Profit for Apr: 4622.0
```

Programming Activity (30 Minutes)

Programming Activity

- Please open 04-practical.ipynb
- Please **ONLY** answer questions 1-5
- We will go through the solutions after 30 minutes

Solutions

SOLID

Single-Responsibility Principle (SRP)

- A class should only have one reason to change
- Responsibilities become coupled if a class has more than one responsibility
- What is a responsibility? “a reason for change”

Open-Closed Principle (OCP)

- Classes, modules & functions should be open for extension, but closed for modification
- Open for extension - an entity's behaviour can be extended
- Closed for modification - extending an entity’s behaviour should not result in changes to its source code

Liskov Substitution Principle (LSP)

- Subtypes must be substitutable for their base types
- If S is a subtype of T then objects of type T may be replaced with objects of type S

Interface Segregation Principle (ISP)

- Clients should not be forced to depend on methods that they do not use
- Results in accidental coupling between clients

Dependency Inversion Principle (DIP)

- High-level modules should not depend on low-level modules. Both should depend on abstractions
- Abstractions should not depend on details. Details should not depend on abstractions