Splitting and joining string

```
string = "Steven said: Hello World"
string.split(": ")
# Result: ['Steven said', 'Hello World']

sample_list = ['Steven said', 'Hello World']
": ".join(sample_list)
# Result: 'Steven said: Hello World'
```

Different approaches to store list of quotations

In this quotes code example, we explore different approaches to store a list of quote record with *who* saying *what*.

Using List and tuple

```
quotes = (
    "Life is 10% what happens to you and 90% how you react to it.—
Charles R. Swindoll",
    "Be brave enough to live life creatively. The creative place where
no one else has ever been.—Alan Alda",
    "The secret of getting ahead is getting started.—Mark Twain"
)
```

Pros:

- Simple data structure

Cons:

- Column separation relies on the content. If there is more than one dash in the string, the logic breaks.

Using Nested list/tuple

```
quotes = (
    ("Life is 10% what happens to you and 90% how you react to it.",
"Charles R. Swindoll"),
    ("Be brave enough to live life creatively. The creative place where
no one else has ever been.", "Alan Alda"),
    ("The secret of getting ahead is getting started.", "Mark Twain")
)
```

Pros:

- We often meet this format when reading CSV or tabular data from external sources/files.
- Easy to use
- Separation not related to the content.

Cons:

- Correctness depends on the order of the list items.

Using Dictionary

```
quotes = (
    {"quote":"Life is 10% what happens to you and 90% how you react to
it.", "source":"Charles R. Swindoll"},
    {"quote":"Be brave enough to live life creatively. The creative
place where no one else has ever been.", "source":"Alan Alda"},
    {"quote":"The secret of getting ahead is getting started.",
"source":"Mark Twain"}
)
```

Pros:

- Using key instead of order to improve code readability
- Easier to maintain

Cons:

- A typo in the key may cause error that is not easily identifiable.

Using namedtuple

```
from collections import namedtuple

Quote = namedtuple("Quote", "content, source") # Quote is upper case

quotes = (
   Quote(content="Life is 10% what happens to you and 90% how you react
to it.", source="Charles R. Swindoll"),
   Quote(content="Be brave enough to live life creatively. The creative
place where no one else has ever been.", source="Alan Alda"),
   Quote(content="The secret of getting ahead is getting started.",
source="Mark Twain")
)
```

Pros

- The column names are pre-defined, so the usage is predictable.
- Typos will raise error at runtime, so it is easy to be identified.
- Using column name instead of order to improve code readability

Cons

- Less flexible than the previous solutions

Randomly pick an item from list

```
import random

quote = random.choice(quotes)
```

Reading and Writing plain text files

We can use `open()` to read and write plain text file. There are 3 modes when opening a file:

- `r` for reading.
- `w` for over-writing.
- `a` for appending.

Reading plain text file

```
with open("quotes.txt") as file_obj:
    quotes = file_obj.read().splitlines()

print(quotes)
```

Reading CSV file

```
import csv

with open("quotes.csv") as file_obj:
    csv_reader = csv.reader(file_obj)
    for line in csv_reader:
        print(line)
```

Writing plain text file

```
with open('text.txt', "a") as file_obj:
    file_obj.write("Hello plain text file.\n")
```

Code Example: Writing plain text

```
import datetime

content = input("What do you want to say to Mr. Diary? ")
if len(content) > 0:
    with open('diary.txt', "a") as file_obj:
        today = datetime.date.today().isoformat()
        file_obj.write(f"{today}: {content}\n")
```

Getting current date in YYYY-MM-DD format

```
import datetime

today = datetime.date.today().isoformat()
```

Getting help with help, dir, or ?

- `help(datetime)`
- `dir(datetime)`
- `datetime?` *Only in Jupyter notebook.*

Reading .docx file

We can use python-docx module to write content to DOCX. We can install extra library by using **`pip install python-docx`**.

```
import docx
doc = docx.Document("Sample Document.docx")
for paragraph in doc.paragraphs:
    print(paragraph.text)
```

Writing .docx file

Code example: Taking input and write to DOCX file

```
import datetime
import docx
import os

content = input("What do you want to say to Mr. Diary? ")
if len(content) > 0:
    with open('diary.txt', "a") as file_obj:
        today = datetime.date.today().isoformat()
        file_obj.write(f"{today}: {content}\n")

if os.path.isfile("diary.docx"):
    doc = docx.Document("diary.docx") # open existing file.
else:
    doc = docx.Document()             # create a new doc.

doc.add_paragraph(content)
doc.save("diary.docx")

print(f"{content} is written to diary.docx")
```