Olio-ohjelmointi 6

Generaattorit

Generator on erityinen tapa luoda **iteroitavia olioita**, jotka tuottavat arvoja **laiskasti (lazy evaluation)** eli yksi kerrallaan, sen sijaan että kaikki arvot laskettaisiin ja tallennettaisiin muistiin kerralla.

Keskeiset piirteet:

- Generaattori **ei palauta koko listaa**, vaan tuottaa arvot sitä mukaa kun niitä tarvitaan.
- Toteutetaan yleensä `yield` -avainsanalla funktiossa.
- **Muistitehokas**: ei varaa koko sarjaa muistiin.
- **Iteroitava**: toimii `for` -silmukassa kuten lista, mutta arvot lasketaan dynaamisesti.

Esimerkki:

```
def count_up_to(n):
    i = 1
    while i <= n:
        yield i  # palauttaa arvon ja jatkaa seuraavasta kohdasta
        i += 1

for num in count_up_to(5):
    print(num)
```

Generaattorin edut:

- **Muistitehokkuus**: Hyvä suurille datamäärille.
- **Laiska laskenta**: Arvot lasketaan vasta kun niitä tarvitaan.
- **Helppo luoda**: `yield` tekee koodista selkeää.

Generaattorit verrattuna listoihin:

- Lista: kaikki arvot tallennetaan muistiin.
- Generaattori: arvot tuotetaan yksi kerrallaan.

Toinen tapa luoda generaattori on käyttää **generaattorilauseketta (generator expression)**, joka muistuttaa listan comprehensiota (list comprehension):

```
squares = (x * x for x in range(10))  # generaattorilauseke

for square in squares:
    print(square)
```

Enum

Enum (lyhenne sanasta "enumeration") on luokka, joka tarjoaa tavan määritellä joukko nimettyjä vakioita. Enum-luokat ovat hyödyllisiä, kun halutaan ryhmitellä liittyviä vakioita yhteen paikkaan ja tehdä koodista luettavampaa ja helpommin ylläpidettävää. Enum-arvot ovat muuttumattomia, mikä tarkoittaa, että niiden arvoja ei voi muuttaa luomisen jälkeen.

Enum-luokat määritellään käyttämällä `enum`-moduulia.

```
from enum import Enum

class Colour(Enum):
    RED = 1
    GREEN = 2
    BLUE = 3
    YELLOW = 4
    BLACK = 5
    WHITE = 6

    def is_primary(self):
        return self in (Colour.RED, Colour.GREEN, Colour.BLUE)

print(Colour.RED)        # Colour.RED
print(Colour.RED.name)   # RED
print(Colour.RED.value)  # 1
```

Enum-instanssia voidaan verrata toisiaan koska jokaisella on oma arvo:

```
print(Colour.RED == Colour.GREEN) # False
```

Enum-luokka voidaan käydä läpi saamaan mahdolliset arvot/instanssit:

```
for colour in Colour:
    print(colour)
```

Enum-instassia voidaan saada myös attribuuttien nimellä tai arvolla:

```
colour = Colour['RED']          # Index-muoto saamaan instanssin nimen perustella
print(colour)                   # Colour.RED

colour = Colour(1)              # Funktio/kutsuminen-muoto saamaan instanssin arvon perustella
print(colour)                   # Colour.RED
```

Enum on Python luokka kuitenkin ja voi sisältää metodeja:

```
print(colour.is_primary())          # True (RED)
print(colour.YELLOW.is_primary())   # False
```

On myös mahdollista määrittää arvoja automaattisesti `auto` -funktiolla:

```
from enum import Enum, auto

class Colour(Enum):
    RED = auto()
    GREEN = auto()
    BLUE = auto()
    YELLOW = auto()
    BLACK = auto()
    WHITE = auto()
```

Voidaan myös määrittää Enumia suoraan Enum-luokasta:

```
Colour = Enum('Colour', [('RED', 1), ('GREEN', 2), ('BLUE', 3)])
```

`Enum` -luokan lisäksi `enum` -moduuli tarjoa eri luokkaa kuten `IntEnum`, `StrEnum`, `IntFlag`...

https://docs.python.org/3/library/enum.html

Lisää aiheesta: https://realpython.com/python-enum/