Python Tips #2

Wim R.M. Cardoen
Email: $(prefix)[at]gmail[dot]com
where
prefix='wcardoen'

August 2, 2022

In the following paragraphs we discuss some recent innovations in Python. All code snippets were tested using Python 3.11.0b3. The examples can be downloaded from the following location:
https://github.com/wcardoen/python-reflections.

- Match-operator:
  A lot of programming languages have selection control mechanisms beyond the `if`, `elif`, `else` constructs. Its C counterpart is the `switch` construct.

```
# Traditional if/elif/else statements
def find_capital(country):
    if country == 'France':
        return 'Paris'
    elif country == 'Germany':
        return 'Berlin'
    elif country == 'Netherlands':
        return 'Amsterdam'
    elif country == 'Belgium':
        return 'Brussels'
    else:
        return 'SORRY!'

for country in ['Belgium', 'Poland']:
    print(f"  Country:{country:>15s} -> Capital:{find_capital(country):>10s}
       ")
```

  The aforementioned code block results in the following output:

```
Country:        Belgium -> Capital:  Brussels
Country:         Poland -> Capital:    SORRY!
```

  In Python 3.10, the `match` construct was introduced.

```
def find_capital2(country):
# match/case construct (Python >= 3.10)
    match country:
        case 'France':
            return 'Paris'
        case 'Germany':
            return 'Berlin'
        case 'Netherlands':
            return 'Amsterdam'
        case 'Belgium':
            return 'Brussels'
        case _:
            return 'Sorry!'

for country in ['Belgium', 'Denmark']:
    print(f"  Country:{country:>15s} -> Capital:{find_capital2(country):>10s
       }")
```

The aforementioned block results into:

```
  Country:        Belgium -> Capital:  Brussels
  Country:        Denmark -> Capital:    Sorry!
```

You can combine several patterns using the | (i.e. ∪ operator).

```
def find_continent(country):
    match country:
        case 'Belgium'|'France'|'Germany'|'Netherlands':
            return 'Europe'
        case 'China'|'India'|'Japan':
            return 'Asia'
        case _:
            return 'Sorry!'
for country in ['France','China','USA','India']:
    print(f"  Country:{country:>15s} -> Continent:{find_continent(country)
       :>10s}")
```

The above code block produces the following output:

```
  Country:         France -> Continent:    Europe
  Country:          China -> Continent:      Asia
  Country:            USA -> Continent:    Sorry!
  Country:          India -> Continent:      Asia
```

Patterns can also be verified by unpacking:

```
def find_location(point):
    match point:
        case (0,0,0):
            return "Origin"
        case (x,0,0):
            return "Pt. on x-axis"
        case (0,y,0):
            return "Pt. on y-axis"
        case (0,0,z):
            return "Pt. on z-axis"
        case (x,y,0):
            return "Pt. in the xy-plane"
        case (x,0,z):
            return "Pt. in the xz-plane"
        case (0,y,z):
            return "Pt. in the yz-plane"
        case (x,y,z):
            return "Reg. pt."
        case _:
            return "NOT a 3D-point"
for item in [(3,4,5), [2,0,0], (0,0,0), (0,3,2), (3,4,5,6)]:
    print(f"  Pt.:{str(item):>15s}   Type:{find_location(item)}")
```

This results into:

```
  Pt.:      (3, 4, 5)   Type:Reg. pt.
  Pt.:      [2, 0, 0]   Type:Pt. on x-axis
  Pt.:      (0, 0, 0)   Type:Origin
  Pt.:      (0, 3, 2)   Type:Pt. in the yz-plane
  Pt.:   (3, 4, 5, 6)   Type:NOT a 3D-point
```

The match pattern construct is wide topic. Three PEPS (PEP-622, PEP-634, PEP-636) were written to address it.

- Merging of dictionaries
  The merging and update of Python dictionaries has been improved in Python 3.9 by introducing the `|` and `|=` operators. The details are discussed in PEP-0584

```
capitals1 = {'france':'paris', 'germany':'berlin'}
capitals2 = {'france':'paris', 'belgium':'brussels'}

# Merging: Creation of a new dict object
capitals3 = capitals1 | capitals2
print(f"  capitals3:\n{capitals3}")

# Update in-place operation
capitals1 |= capitals2
print(f"  capitals1:\n{capitals1}")
```

```
  capitals3:
{'france': 'paris', 'germany': 'berlin', 'belgium': 'brussels'}
  capitals1:
{'france': 'paris', 'germany': 'berlin', 'belgium': 'brussels'}
```

- Removing the prefixes/suffixes of strings
  Python 3.9 introduced some handy methods to remove suffixes and prefixes (PEP-0616).

```
city="Witwatersrand"
STR1, STR2 ="Wit", "rand"
print(f"String:'{city}'")
print(f"  remove the prefix '{STR1}'  ->  '{city.removeprefix(STR1)}'")
print(f"  remove the sufffix '{STR2}'  ->  '{city.removesuffix(STR2)}'")
```

```
String:'Witwatersrand'
  remove the prefix 'Wit'  ->  'watersrand'
  remove the sufffix 'rand'  ->  'Witwaters'
```

- `math` module
  The `math` module was extended with some interesting methods, among them:
  - `math.isqrt` : returns the integer part of the square root

- `math.gcd` : returns the Greatest Common Divisor
- `math.lcm` : returns the Least Common Multiple
- `math.prod` : calculates the products of the elements in a iterable
- `math.comb` : calculates the number of combinations $\binom{n}{k}$
- `math.perm` : calculates the number of permutations $\frac{n!}{(n-k)!}$
- `math.dist` : calculates the euclidean distance between $p, q \in \mathbb{R}^n$

```
import math
print(f"  math.isqrt(26)    :{math.isqrt(26)}")
print(f"  math.gcd(24,12,36):{math.gcd(24,12,36)}")
print(f"  math.lcm(2,4,6,8) :{math.lcm(2,4,6,8)}")
print(f"  math.prod([2,3,4,5,6],start=10):{math.prod([2,3,4,5,6],start=10)}"
    )
print(f"  math.comb(5,2):{math.comb(5,2)}")
print(f"  math.perm(5,2):{math.perm(5,2)}")
p = range(1,10)
q = range(2,11)
print(f"  math.dist(p,q):{math.dist(p,q)}")
```

which results into:

```
  math.isqrt(26)    :5
  math.gcd(24,12,36):12
  math.lcm(2,4,6,8) :24
  math.prod([2,3,4,5,6],start=10):7200
  math.comb(5,2):10
  math.perm(5,2):20
  math.dist(p,q):3.0
```