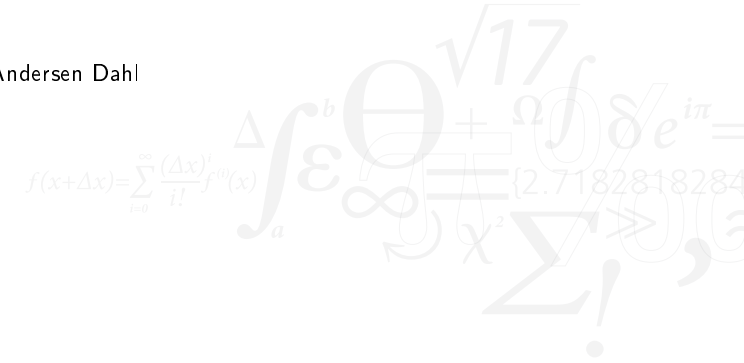
Lecture 7: Tuples

Morten Rieger Hannemose, Vedrana Andersen Dahl

Fall 2023



Today's lecture

1. Tuples (ca. 30 min)
2. Coding example (ca. 30 min)

Practical information

- Mid-term evaluation. Available on DTU Learn.
- Shift in exercises. From now on, some exercises will be more general, to practice combining everything you have learned up to now.
- Coding examples. The code used in the coding examples is available in the last slide.
- Half length test-exam on the 26^{th} of October, with five questions.

Tuples: creation

```
my_list = ['first', 2, '3', 4.0]
my_tuple = ('first', 2, '3', 4.0)
another_tuple = 'Nicolas', 'Cage'

# unpacking
a, b, c, d = my_tuple
A, B, C, D = my_list
```

Re-visit variable swap

```
a = 14.5
b = 3.4

a, b = b, a
```

- Look back at lists: a *mutable* sequence of values indexed by integers starting with 0.
- Tuple – an *immutable* sequence of values indexed by integers starting with 0.
- *Unpacking* is concise Python syntax for extracting all elements from list or tuple into separate variables.
- Word `tuple` is a built-in type, don't use it as a variable name.

Tuples: accessing values

```
1  my_tuple = ('a', 'banana', 17, [1, 2, 3])
2  print('banana' in my_tuple)
3
4  print(my_tuple)
5  print(my_tuple[1])
6
7
8  for element in my_tuple:
9      print(element)
10
11 # my_tuple[0] = 'apple'  # tuple does not
       support item assignment
12 my_tuple[3][0] = 5
13 my_tuple[3].append(4)
14
15 my_tuple += (5, 6) # reassigning the tuple,
        not modifying it
16 print(my_tuple)
```

- Keyword `in` with `for` loop traverses tuple elements

Tuples as return values

```
1  def get_fractions(a):
2      half = a / 2
3      quarter = a / 4
4      eighth = a / 8
5      return half, quarter, eighth
6
7  fractions = get_fractions(10)
8  print(fractions)
9  nr = 24
10 a, b, c = get_fractions(nr)
11 print('Fractions of', nr, 'are:', a, b, c )
```

- In line 5: Return value is a tuple.
- In line 10: Tuple is returned and unpacked.

Advanced syntax regarding variable-length argument tuples Section 12.4 in the book.

- When writing function, gather arguments using `*args`.
- When using a function, scatter a tuple using `*my_tuple`.

Random numbers

```
1 import random
2
3 my_list = ['zero', 'one', 'two', 'three',
4     'four', 'five', 'six', 'seven']
5
6 for i in range(12):
7     x = random.randint(0, len(my_list) - 1)
8     print(my_list[x])
```

- Pseudorandom number generator.
- Various distribution: uniform float in a certain range, uniform int in a certain range, normal distribution, random choice ...

Example 1

Write a function that takes as input an integer number between 0 and 9999. The function should return a 4-element tuple, containing the 4 integers, the digits of the number. Numbers smaller than 1000 should be zero-padded to 4 digits.
Consider how to implement the solution:

- Computation: utilizing properties of division and modulus.
- Strings: utilizing string representation of the integer.

Note:

- Different implementations may lead to the same (correct) solution. Implementation is a way of solving the problem. Still, some implementations may be better than others.
- Very similar to the concept of data selection: should I use lists, tuples, dictionaries...? Discussion in Section 13.9 in the book.

Coding example

`standardize_address.py`, exam from June 2022.

Standardize address

You need to combine addresses from multiple sources, with different ways of writing the address line which includes the city name and the postal (zip) code. The differences are:

- Some write postal code first (like for example `2840 Holte`), while others write city first (for example `Holte 2840`).
- Some use a space to separate the parts of the address (like in the examples above), while others use underscore (like for example `2840_Holte`.

You want to standardize the addresses such that the postal code always comes first, and that space is always used for separating the parts of the address.

Problem formulation

Write a function `standardize_address` which takes as input an address line, which may lack standardization. The function should return the standardized address line. The city name may consist of multiple words, but the city name only consists of letters. The postal code may be of different length, but it only consists of digits. The postal code is always either at the beginning or at the end of the address line.

Consider `New York 10001`. It uses spaces, so it requires no change in the word separator. However, the postal code is at the end after the city name, so it needs to be moved to the beginning. The function should return `10001 New York`.

Code used for coding examples

Example 1

```
def digitize_1(nr):

    d1 = nr % 10
    d2 = (nr % 100) // 10
    d3 = (nr % 1000) // 100
    d4 = (nr % 10000) // 1000
    return d4, d3, d2, d1

def digitize_2(nr):

    d = str(nr)
    d = '0'*(4-len(d)) + d
    return int(d[0]), int(d[1]), int(d[2]),
     int(d[3])

print(digitize_1(106))
print(digitize_2(106))
```

Example 2

```
def standardize_address(addr):
    addr = addr.replace('_', ' ').split()

    if addr[-1].isnumeric():
        addr = addr[-1:] + addr[:-1]
    return ' '.join(addr)

print(standardize_address("New 10001"))
```