Part 7

Introduction to Programming

17.10.2024

Last week

- Reading files
- CSV files
- Splitting strings
- Writing files
- Validating inputs
- Exceptions

course score and exams

No lecture next week, the Advanced course start on Oct 31st!

Modules

The standard library of Python contains several useful modules

We have earlier used for example module **math** to calculate square roots

Using modules

Modules are imported with the **import** statement

```
import math

# luvun 5 neliöjuuri
print(math.sqrt(5))
# luvun 8 logaritmi (2-kantainen)
print(math.log(8, 2))
```

Importing individual operations

Individual routines can be imported with **from module import operation** structure

```
from math import sqrt, log

print(sqrt(5))
print(log(5,2))
```

All routines can be imported with asterisk. This should however be avoided

```
from math import *

print(sqrt(5))
print(log(5,2))
```

Module contents

Standard library modules are described in Python documentation

```
import math

print(dir(math))
```

Operations can be listed with the function **dir**

Random numbers

Module **random** can be used to generate pseudo random numbers

```
from random import randint

for i in range(10):
    print("Dice throw: ", randint(1,6))


from random import shuffle

fruits = ["apple", "banana", "orange", "grapefruit"]
shuffle(fruits)
print(fruits)

from random import choice
fruits = ["apple", "banana", "orange", "grapefruit"]
print(choice(fruits))
```

Handling times

Whenever times and dates are handled, it is a good idea to use the standard library

Things like leap years make handling dates quite complicated otherwise

Datetime objects

Time can be modelled by creating an object from the **datetime** object

```
from datetime import datetime

aika = datetime.now()
print(aika)
```

```
from datetime import datetime

aika = datetime(1952, 12, 24)
print(aika)
```

Comparing times

Datetime objects can be compared with standard comparison operators

```
from datetime import datetime

now = datetime.now()
midsummer = datetime(2020, 6, 20)

if now < midsummer:
    print("It's not midsummer yet")
elif now == midsummer:
    print("Happy midsummer!")
else:
    print("Midsummer is gone")
```

Timedelta object

Comparing times returns a **timedelta** type object

The same object can be used to "roll" time backwards or forwards

```
from datetime import datetime, timedelta

now = datetime.now()
midsummer = datetime(2020, 6, 20)

diff = midsummer - now

print(f"It's {diff.days} days to midsummer" )

week = timedelta(days=7)
midsummer += week

print("Week after midsummer it's", midsummer)
```

Formatting times

Datetime object can be formatted with the **strftime** function

```
from datetime import datetime

aika = datetime.now()
print(aika.strftime("%d.%m.%Y"))
```

Symbol	**Meaning**
%d	Day (01-31)
%m	Month (01-21)
%Y	Year (YYYY)
%H	Hour (24h format)
%M	Minutes 00 to 59
%S	Seconds 00 to 59

Reading CSV files

Although CSV files are easy to handle as text, utilizing the dedicated module can be even easier

```
import csv

with open("testi.csv") as tiedosto:
    for rivi in csv.reader(tiedosto, delimiter=";"):
        print(rivi)
```

Module **csv** is great for this

Reading web pages

In Python, reading a web page (as HTML) is quite wasy

```
import urllib.request

pyynto = urllib.request.urlopen("https://helsinki.fi")
print(pyynto.read())
```

Creating own modules

Any Python program that contains functions can be used as a module

For example **words.py**

```
def first_word(sentence: str):
    parts = sentence.split(" ")
    return parts[0]

def last_word(sentence: str):
    parts = sentence.split(" ")
    return parts[-1]

def number_of_words(sentence: str):
    parts = sentence.split(" ")
    return len(parts)
```

Own module (2)

```
def first_word(sentence: str):
    parts = sentence.split(" ")
    return parts[0]

def last_word(sentence: str):
    parts = sentence.split(" ")
    return parts[-1]

def number_of_words(sentence: str):
    parts = sentence.split(" ")
    return len(parts)
```

```
import words

s = "Hello all it's good to meet you"
print(words.first_word(s))
print(words.last_word(s))
print(words.number_of_words(s))
```

Main program

If module contains program code outside functions, it is always executed when the module is imported

This can be prevented by including code inside `if __name__ == "__main__"` block

Some useful techniques

In Python, simple if-else conditions can be written with as a single expression

```
if x % 2 == 0:
    print("Even")
else:
    print("Odd")


print("Even" if x % 2 == 0 else "Odd")
```

Pass

Statement **pass** does not do anything

```
def testi():
    pass
```

It is still useful, as Python does not allow empty blocks

Default parameter value

Function parameters can be provided default values

In such cases, the argument may be omitted when calling the function

```
def tervehdi(nimi="Emilia"):
    print("Moikka,", nimi)

tervehdi()
tervehdi("Erkki")
tervehdi("Matti")
```

Changing number of parameters

The parameter number can be defined as changing by typing an asterisk in front of parameter name

```
def test(*values):
    print(f"You gave {len(values)} arguments" )
    print(f"The sum is {sum(values)}" )


test(1,2,3)
test(10,20,30,40,450,60)
```

Advanced course

Object oriented programming

Functional programming

Programming games

About exam

Duration 4 hours (or 5, if extra time allowed)

You can start any time between 10 and 22. Please note, that the exam ends at 22 the latest regardless of when you started.

3 to 4 coding tasks

Exercises are not too difficult

Assessed manually, small errors don't count

Example question

"Write a program which queries the user for strings, one at a time. The program then proceeds to output how many strings were given, which was the longest, how long it was and what was the most frequently used character"

…including example execution etc.