1. Úloha `02_MOCK`

Základní informace:

- **Účel:** prakticky se seznámit s možnostmi parametrizace jednotkových testů a s technikou využití *mock* objektů
- **Odevzdávané soubory:**

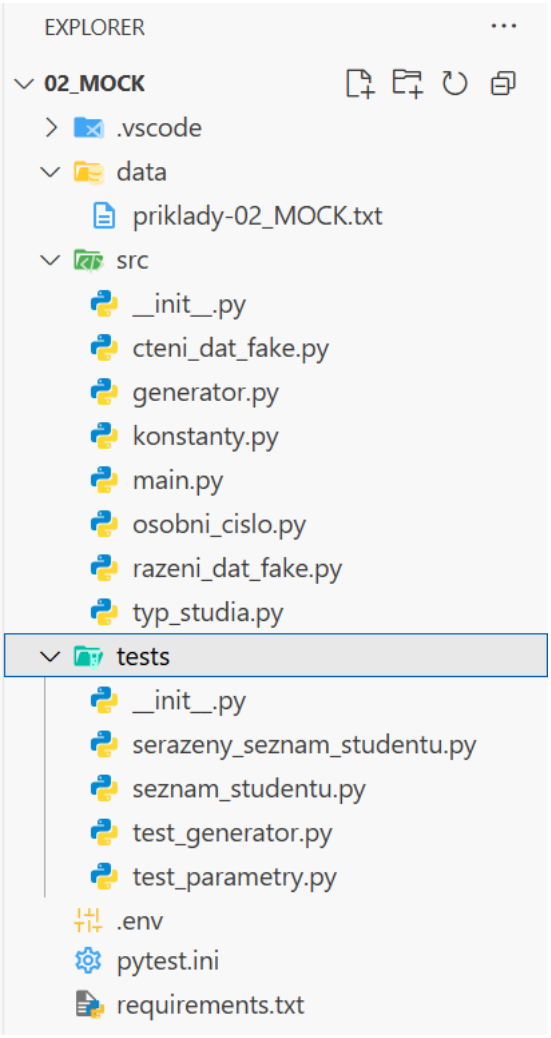
```
serazeny_seznam_studentu.py
seznam_studentu.py
test_generator.py
test_parametry.py
```

Zadání:

- otestujte doménovou třídu jednotkovými testy s využitím *mock* objektů
- připravte parametrizované jednotkové testy, tj. využijte pokročilé možnosti jednotkových testů

Popis vstupních dat:

- adresář úlohy



- v adresáři `data` se nachází datový soubor `priklady-02_MOCK.txt`

- data pro generování osobního čísla
  - každý řádek je rozšířen o jeden údaj, který představuje očekávaný výsledek testu metody `OsobniCislo.is_platne_osobni_cislo()`
  - pro účely ověření správnosti je na několika místech tento údaj zadán úmyslně chybně
- tento soubor budete v parametrizovaném testu pouze číst a nebudete jej nijak měnit

■ v adresáři `src` se nachází zdrojové `.py` soubory

- adresář obsahuje následující soubory:
  - `osobni_cislo.py` a třída `OsobniCislo`
    - narozdíl od úlohy `01_UNIT` je tato třída již bez defektů a budete ji považovat za plně funkční
    - tento soubor nebudete měnit, ale budete jej testovat
  - `konstanty.py` a třída `Konstanty`
    - tento soubor nebudete ani testovat, ani měnit
  - `typ_studia.py` a výčtový typ `TypStudia`
    - tento soubor nebudete ani testovat, ani měnit
  - `generator.py` a třída `Generator`
    - tento soubor nebudete měnit, ale budete jej testovat
  - `cteni_dat_fake.py` a třída `CteniDat`
    - tento soubor nebudete ani testovat, ani měnit
    - třída `CteniDat` slouží jako simulovaný datový zdroj při spuštění `main.py`
      - je to praktická ukázka využití *fake* objektů
  - `razeni_dat_fake.py` a třída `RazeniDat`
    - tento soubor nebudete ani testovat, ani měnit
    - třída `RazeniDat` slouží jako simulovaný datový zdroj při spuštění `main.py`
      - je to praktická ukázka využití *fake* objektů
  - `main.py` - triviální spouštěcí skript, který ukazuje funkčnost tříd `OsobniCislo` a `Generator`
    - tento soubor nebudete ani testovat, ani měnit

■ v adresáři `tests` se nacházejí soubory pro jednotlivé druhy testů

- adresář obsahuje následující soubory:
  - `test_generator.py` - kostra testů pro ukázku mockování

  - tento soubor budete doplňovat
- `test_parametry.py` - kostra parametrizovaných testů
  - tento soubor budete doplňovat
- `seznam_studentu.py` - seznam studentů pro mockování
  - tento soubor můžete libovolně doplňovat
- `serazeny_seznam_studentu.py` - abecedně seřazený seznam studentů z `seznam_studentu.py` pro mockování
  - tento soubor musíte doplňovat v souladu s obsahem souboru `seznam_studentu.py`

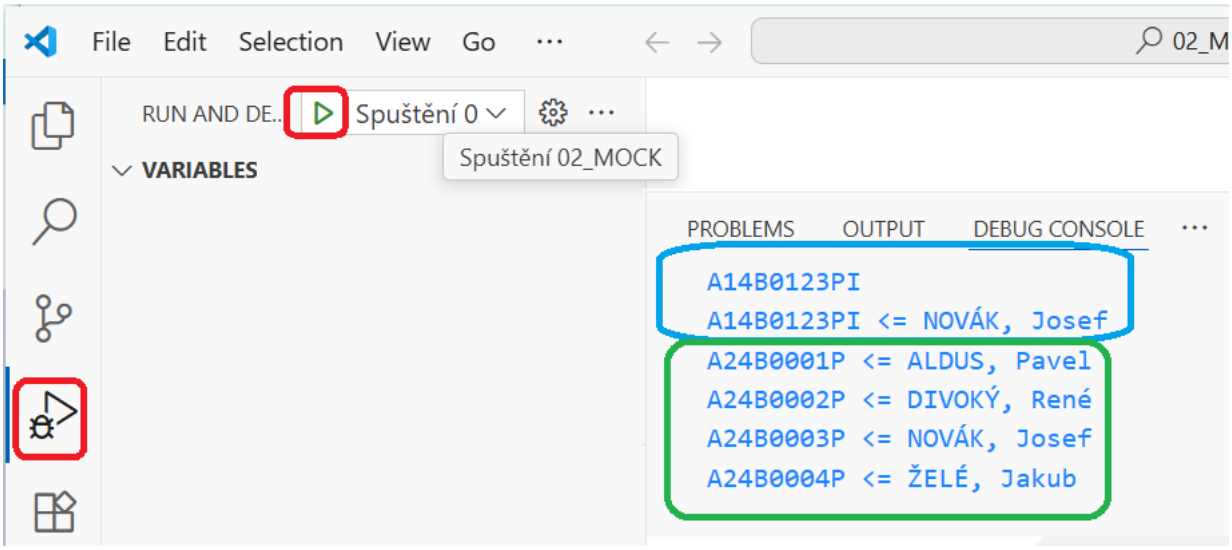
- v kořenovém adresáři se nachází soubory:
  - `.env` - konfigurační soubor prostředí
    - tento soubor nebudete měnit
  - `pytest.ini` - konfigurační soubor spouštění **pytest**
    - tento soubor nebudete měnit

Postup řešení:

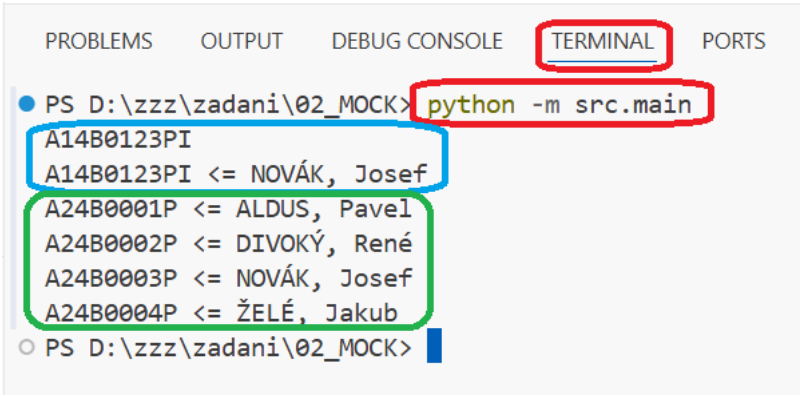
- budete řešit dvě oddělené úlohy
  - test s využitím mockování (`test_generator.py`) - testujete pouze třídu `Generator`
  - parametrizovaný test (`test_parametry.py`) - testujete pouze třídu `OsobniCislo`

Test s využitím mockování

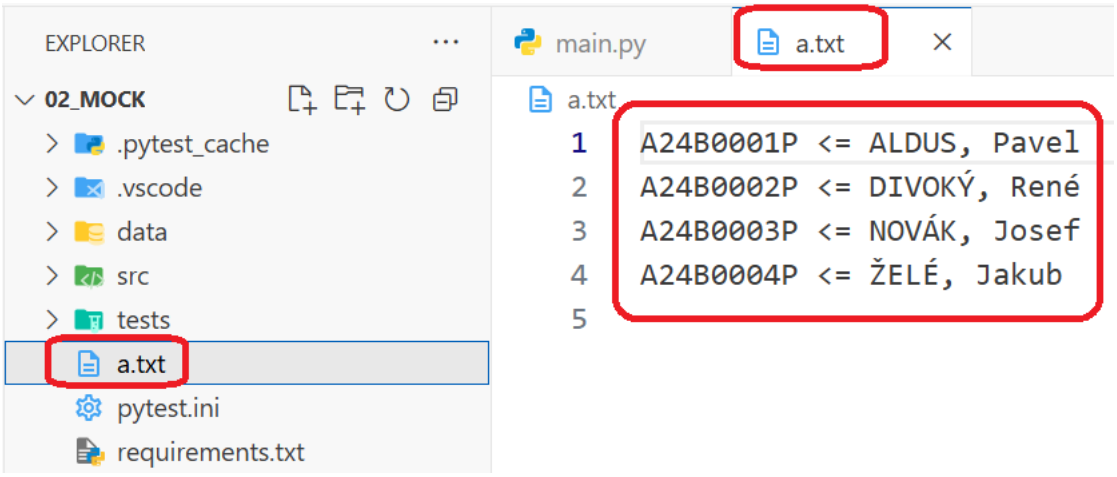
- můžete kontrolně spustit `main.py` pro ujištění, že třída `Generator` funguje správně
  - modře orámovaný výpis je ukázka, že třída `OsobniCislo` je již plně funkční
    - můžete ji ze zájmu kontrolně srovnat s její verzí z úlohy `01_UNIT`
  - zeleně orámovaný výpis je ukázka činnosti třídy `Generator`



- spuštění z terminálu příkazem `python -m src.main`



- třída `Generator` vytváří soubor `a.txt`



- správná funkcionalita třídy `Generator` závisí na třídách `CteniDat` a `RazeniDat`
  - obě jsou zde nahrazeny *fake* objekty
  - toto je jen ukázka použití *fake* objektů, se kterými ale nebudete dále nijak pracovat

main.py

src > main.py > ...

```
5   from osobni_cislo import OsobniCislo
6   from generator import Generator
7   from cteni_dat_fake import CteniDat
8   from razeni_dat_fake import RazeniDat
9
10  def main() -> None:
11      oc: OsobniCislo = OsobniCislo("Novák, Josef, fav, 2014, b,
12      print(oc.vysledek)
13      print(oc.__str__())
14      generator: Generator = Generator(CteniDat(), RazeniDat())
15      seznam: list[OsobniCislo] = generator.generuj_seznam_osobn
16                                      fakulta="fav", rok=2024, typ_studi
```

- třída `Generator`
  - její jediná metoda `generuj_seznam_osobnich_cisel()` by se v praxi používala pro generování osobních čísel na základě seznamu příjmení a jmen studentů příslušného ročníku a studijního programu
  - svůj výstup zapisuje primárně do souboru
    - ten by následně použila studijní referentka pro import do STAGu
    - výstupní soubor nás v této úloze nebude zajímat - můžete si jej ale pro kontrolu ověřit - viz výše soubor `a.txt`
  - seznam osobních čísel vygenerovaných metodou `generuj_seznam_osobnich_cisel()` je pro lepší testovatelnost metody vracen jako návratová hodnota
  - *pseudozdůvodnění, proč použít mockování:*
    - metoda `generuj_seznam_osobnich_cisel()` ke své správné činnosti potřebuje dvě další metody, které ale nejsou momentálně k dispozici a pro otestování metody `generuj_seznam_osobnich_cisel()` budou muset být mockovány:
      - `nacti_studenty()` - získání seznamu příjmení a jmen studentů
        - k dispozici není proto, že se uvažuje o získávání tohoto seznamu přes RestAPI přímo ze STAGu
      - `serad_podle_abecedy()` - seřazení seznamu podle abecedy
        - k dispozici není proto, že řazení podle české normy je mimořádně komplikované (tři úrovně "řadicí platnosti") - `https://prirucka.ujc.cas.cz/?id=885`
        - tyto požadavky nesplňuje běžné řazení poskytované Pythonem
          - je nutné importovat speciální modul `icu`, který ale potřebuje ICU knihovnu (*International Components for Unicode*) a nástroj `pkg-config`, který je obvykle dostupný jen v unixových prostředích

- pro ověření správné funkce generátoru není třeba provádět speciální importy

- protože budou v testu použity dva mocky, je třeba pro ně připravit datové zdroje
- v adresáři `test` modifikujte skript `seznam_studentu.py` s obsahem např.:

```
STUDENTI = [
    "Novák, Josef",
    "Aldus, Pavel",
    "Želé, Jakub",
    "Divoký, René"
]
```

- dále v adresáři `test` doplňte skript `serazeny_seznam_studentu.py` s obsahem, který bude velmi podobný předchozímu skriptu, jen s tím rozdílem, že příjmení a jména budou seřazena podle abecedy

```
SERAZENI_STUDENTI = [...
```

Note

V reálném testu pomocí mockování by seznamy `STUDENTI` a `SERAZENI_STUDENTI` byly přímo součástí skriptu `test_generator.py`.

Způsob použitý zde ale dává možnost, že ve validátoru budou existovat stejně pojmenované soubory, ovšem s jinými seznamy studentů. Tím bude možné ověřit univerzálnost testu.

Je zaručeno, že seznamy na validátoru nikdy nebudou prázdné.

- obsah skriptu `test_generator.py`
  - doplňte kostru funkce `test_generovani_cisel(mocker)` která bude volat

```
seznam: list[OsobniCislo] = ▶
generator.generuj_seznam_osobnich_cisel(vystup="a.txt", \
                         fakulta="fav", rok=2024, typ_studia="b", ▶
forma_studia="p")
```

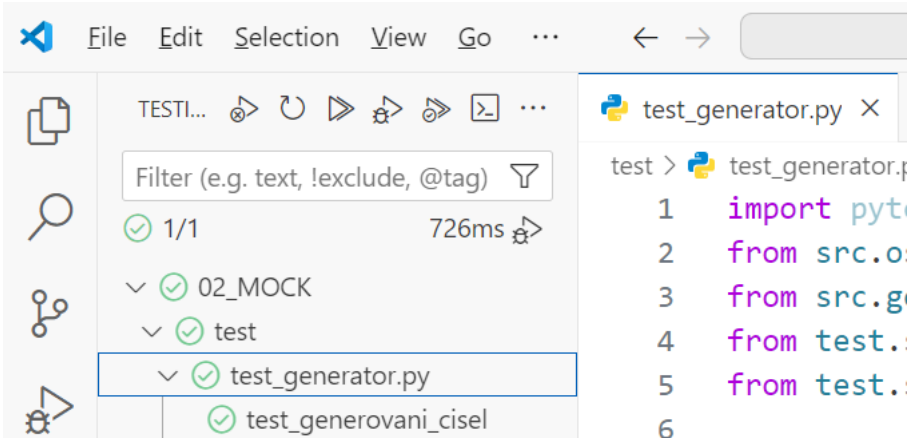
  - na jménu výstupního souboru `a.txt` nezáleží
  - to, že je vrácený `seznam` správně, ověříte dvěma asserty
    - první bude zjišťovat shodnost délky seznamu studentů (ze skriptu `serazeny_seznam_studentu.py`, tj. data předaná do mocku) s délkou `seznam`
    - druhý bude zjišťovat, zda pořadové číslo v osobním čísle posledního studenta získaného voláním `get_osobni_cislo()` (např. `0004` pro předchozí ukázku ze skriptu `seznam_studentu.py`) se shoduje s délkou `seznam`

Note

Při reálném testování by se určitě daly provádět sofistikovanější aserty, ale pro vyzkoušení principu mockování to zde není důležité.

  - na samý závěr testovací funkce ověří, zda byly oba mocky skutečně volány

- **test musí projít** a výsledek by měl být



Parametrizovaný test - obsah skriptu `test_parametry.py`

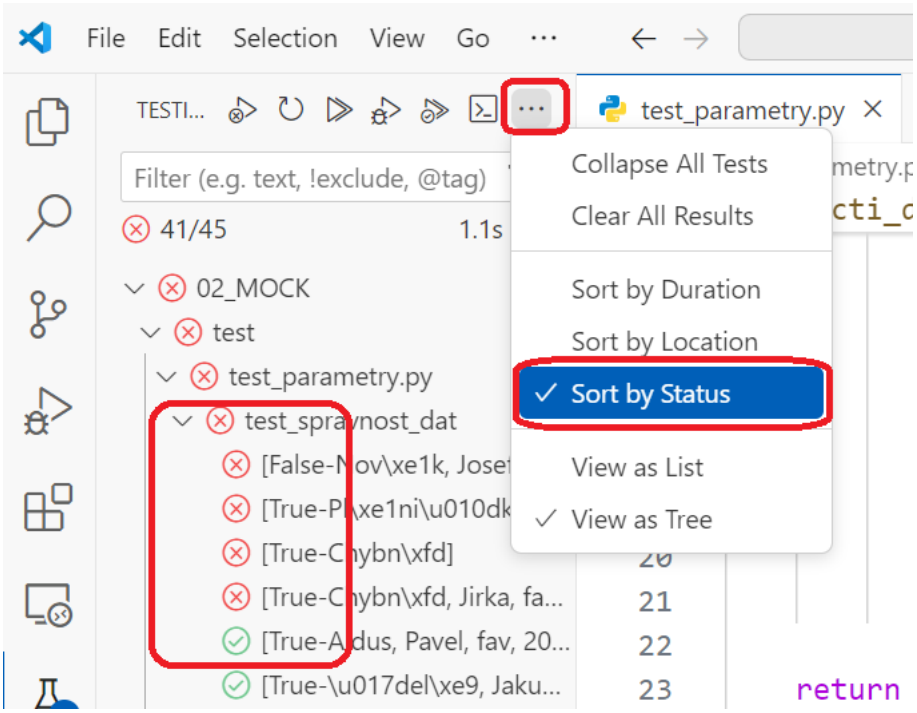
- prohlédněte si obsah souboru `data/priklady-02_MOCK.txt`
  - data v něm budou řídit parametrizovaný test
- doplňte kostru funkce se signaturou

```
def nacti_data_ze_souboru() -> list[tuple[bool, str]]:
```

  - funkce bude sloužit jako datový zdroj testovací funkce
  - v těle funkce neměňte již hotový výpočet cesty k souboru
  - načte soubor `priklady-02_MOCK.txt` do seznamu řádek, ve kterém ponechá jen platné řádky
    - tj. vynechá prázdné řádky a řádky komentářů začínající `#`
  - načtené řádky parsuje podle znaku ;
    - obě části uloží do N-tice (*tupple*)
      - první část má význam `vysledek: bool`
      - druhá část má význam `radka_dat: str`
  - seznam N-tic vrací
- doplňte kostru funkce testující metodu `is_platne_osobni_cislo()` se signaturou

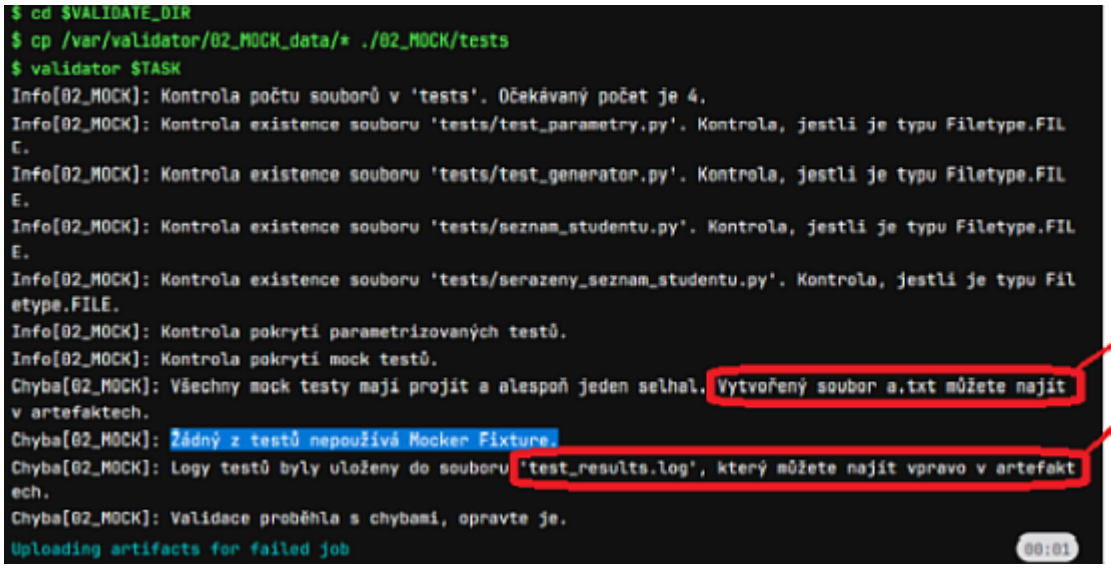
```
@pytest.mark.parametrize("vysledek, radka_dat", nacti_data_ze_souboru())
def test_spravnost_dat(vysledek: bool, radka_dat: str):
```

  - skutečné parametry této funkce budou dodávány z datového zdroje `nacti_data_ze_souboru()`
  - funkce vytvoří instanci osobního čísla pomocí parametru `radka_dat`
  - funkce pak pomocí parametru `vysledek` porovná, zda má vygenerované osobní číslo platný formát
- spusťte test `test_spravnost_dat()`
  - výsledek by měl být



Pomocné informace, když selže validace:

- pokud validační skript nalezne selhání, tj. odevzdávaná úloha nevyhovuje, pak je možné zjistit dodatečné informace o průběhu testu
- jsou uloženy ve artefaktech, ve kterých se proklikáte na dva soubory:
  - `a.txt` - soubor osobních čísel vytvořený generátorem z `test_generovani_cisel()`
  - `test_result.log` - záznam průběhu testů z obou částí (mock i parametrizovaný test) úlohy



Note

Pokud validační skript projde, artefakty se nevytvářejí.

Note

Artefakty se uchovávají jen po velmi omezenou dobu 1 hodiny od svého vzniku.

Commitované soubory:

- ```
  serazeny_seznam_studentu.py
  seznam_studentu.py
  test_generator.py
  test_parametry.py
  ```

Časová náročnost úlohy:

- po úspěšném odevzdání, prosím, vyplňte v tabulce časovou náročnost této úlohy

  https://docs.google.com/spreadsheets/d/1brjDkZ4pBkAD1WG7tLan89zFHkOcoaQMQXtcTq1JtU0
  ```