https://www.alx.pl/ankiety/4473wuwi

https://www.linkedin.com/in/gradzinski/

Linki na początek

- http://bit.ly/alx-kpython-formularz
- https://github.com/piotrgradzinski/python_20220124
- https://pl.wikipedia.org/wiki/Metoda_gumowej_kaczuszki
- p.gradzinski@alx.pl
- https://www.markdownguide.org/cheat-sheet/
- https://www.python.org/dev/peps/pep-0008/
- https://www.jetbrains.com/pycharm/download/#section=windows
- Pakiet do sprawdzania dni wolnych od pracy: https://pypi.org/project/holidays/

Przygotowanie środowiska

- **Python**
  - https://www.python.org/downloads/
- **GIT**
  - https://git-scm.com/download/win
  - Wprowadzenie do GIT: https://www.youtube.com/watch?v=BCQHnlnPusY&list=PLRqwX-V7Uu6ZF9C0YMKuns9sLDzK6zoiV&ab_channel=TheCodingTrain
- **PyCharm Community Edition**
  - https://www.jetbrains.com/pycharm/download
- Notepad++ - https://notepad-plus-plus.org/downloads/
- Sublime Text - https://www.sublimetext.com/
- Visual Studio Code - https://code.visualstudio.com/download
  - skróty klawiszowe: https://code.visualstudio.com/docs/getstarted/keybindings

VENV

- https://code.visualstudio.com/docs/python/environments
- ustawiamy domyślny terminal na Command Prompt
  - nowy terminal
  - strzałka obok plusa
  - Select Default Profile
  - wybieramy Command Prompt
- `conda create --name nazwa_srodowiska python=WERSJA_PYTHONA`
- `activate nazwa_srodowiska`

- klikamy na interpreter (lewy dolny róg) i odświeżamy listę i wybieramy nowo dodane środowisko

O pythonie

- https://www.tiobe.com/tiobe-index/
- Guido van Rossum - twórca Pythona
- działa na popularnych systemach operacyjnych: Windows, Mac, Linux
- zastosowania pythona:
    - do pisania skryptów uruchamianych z poziomu konsoli
    - programy okienkowe
    - strony internetowe (framework django, flask)
    - gry
    - aplikacje mobilne
    - przetwarzanie danych
        - obliczenia
        - analiza tekstu
        - AI / Machine Learning
        - data science
- kto używa Pythona
    - youtube
    - google
    - dropbox
    - instagram
    - spotify
    - NASA
    - https://en.wikipedia.org/wiki/List_of_Python_software

Bity i bajty

- https://www.mathsisfun.com/binary-decimal-hexadecimal-converter.html
- https://pl.wikipedia.org/wiki/Systemy_pozycyjne
- https://pl.wikipedia.org/wiki/Dw%C3%B3jkowy_system_liczbowy
- https://alx.piotr.gg/python/system_pozycyjny.pdf

Liczby zmiennoprzecinkowe

- https://ryanstutorials.net/binary-tutorial/binary-floating-point.php
- https://stackoverflow.com/questions/15004944/max-value-of-integer
- http://www.exploringbinary.com/why-0-point-1-does-not-exist-in-floating-point/

Obliczenia na liczbach zmiennoprzecinkowych

- https://zetcode.com/python/decimal/
- https://www.sympy.org/en/index.html

Napisy

- https://realpython.com/python-f-strings/
- https://www.python.org/dev/peps/pep-0498/#format-specifiers
- https://docs.python.org/3.9/library/string.html#formatspec

Operatory

- operatory arytmetyczne, wynikiem ich działania jest int albo float
  - +, -, *, /, //, %, **
- operatory działające z str
  - + - konkatenacja
  - * - powielanie napisu
- operatory porównania, wynikiem ich działania jest bool, czyli True albo False
  - ==, !=, <, >, <=, >=
- operatory przypisania
  - =, +=, -=, *=, /=, //=
- operatory logiczne
  - koniunkcja (logiczne i) - and
  - alternatywa (logiczne lub) - or
  - negacja, zaprzeczenie (logiczne nie) - not
    - not True -> False
    - not False -> True

A	B	A and B	A or B
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

Boolean

- https://realpython.com/python-boolean/#python-boolean-testing

Instrukcja warunkowa if

`if warunek:` - obowiązkowy
`else:` - może wystąpić tylko jedno, nieobowiązkowe
`elif warunek2:` - może wystąpić wiele razy, nieobowiązkowy

```
if warunek:
      zrób jak warunek będzie prawdziwy (True)
```

```
if warunek:
      zrób jak warunek będzie prawdziwy (True)
else:
      zrób jak warunek będzie fałszywy (False)
```

```
if warunek1:
      zrób jak warunek1 będzie prawdziwy (True)
elif warunek2:
      zrób jak warunek2 będzie prawdziwy (True)
else:
      zrób jak wszystkie wcześniejsze warunki będą fałszywe (False)
```

```
if warunek1:
      zrób jak warunek1 będzie prawdziwy (True)
elif warunek2:
      zrób jak warunek2 będzie prawdziwy (True)
elif warunek3:
      zrób jak warunek3 będzie prawdziwy (True)
else:
      zrób jak wszystkie wcześniejsze warunki będą fałszywe (False)
```

```
if warunek1:
      zrób jak warunek1 będzie prawdziwy (True)
elif warunek2:
      zrób jak warunek2 będzie prawdziwy (True)
```

Pętla while

- sprawdzamy warunek
    - jak jest prawdziwy, to wykonujemy ciało pętli
    - jak jest fałszywy, to kończymy wykonywanie pętli
- najpierw sprawdzamy warunek, później wykonujemy ciało pętli

Kolekcje

Tupla, krotka (ang. tuple)

- https://realpython.com/python-lists-tuples/
- https://stackoverflow.com/questions/626759/whats-the-difference-between-lists-and-tuples
- definiujemy w `( ,)`
- tupli po utworzeniu NIE możemy zmienić
- **jest niemutowalna**
- dane wewnątrz tupli są uporządkowane (ang. ordered), tupla zachowuje informację o kolejności, w jakiej włożyliśmy do niej elementy
- operator przypisania nie zadziała z tuplą, dostaniemy błąd TypeError
- Do czego używamy tupli? Używamy do trzymania danych heterogeniczne.

Lista (ang. list)

- definiujemy w `[ ]`
- **jest mutowalna,** czyli możemy dodawać do niej elementy po utworzeniu, albo zmieniać istniejące elementy listy
- podobnie jak tupla lista jest uporządkowana
- Do czego używamy list? Używamy do trzymania danych homogenicznych.

Słownik (ang. dictionary)

- definiujemy w `{ klucz: wartość }`
- **jest mutowalna**, uporządkowany (przed 3.7 nieuporządkowany)
- restrykcje na klucz:
    - prawie wszystkie typy danych z pythona mogą być kluczem
    - można powiedzieć, że typy niemutowalne mogą być kluczem
    - precyzyjnie, tylko te typy danych, które są hashowalne, czyli posiadają metodę __hash__() mogą być używane jako klucz
- wartością może być cokolwiek
- https://realpython.com/python-dicts/

Zbiór (ang. set)

- definiujemy `{ }`
- **jest mutowalny**, nieuporządkowany
- w ramach zbioru przechowujemy tylko unikalne elementy
- na zbiorze możemy wykonywać operacje teoriomnogościowe
- musi zawierać elementy niemutowalne (posiadające zaimplementowaną metodę __hash__(), czyli takie elementy, które mogą być kluczami w słowniku)
- nie możemy używać operatora dostępu `[ ]`
- https://realpython.com/python-sets/

Wyrażenia listowe

- wyrażenia pozwalają nam w prosty sposób tworzyć listy
- jak zdefiniuję je w [ ], to wtedy takie wyrażenie stworzy mi listę elementów
- wewnątrz tego wyrażenia mamy zwykłą pętle for, która ma wypluwać z siebie elementy
- https://realpython.com/lessons/generalized-list-comprehension-structure/

```
(values) = [(expression) for (value) in (collection)]
```

Funkcje

Z matematyki
f(x) = x + 1

```
def nazwa_funkcji (argument1, argument2, ...)
      definiujemy co ta funkcja ma robić
```

- funkcje w pythonie musimy zdefiniować, żeby ją użyć
- do zdefiniowania funkcji używamy słowa kluczowego `def`
- zwyczajowo, nazwy funkcji są zapisywane `snake_case`
- return
    - służy do zwracania wartości z funkcji
    - jeżeli python uruchomi returna, to spowoduje wyjście z funkcji!

Testy

- testy jednostkowe - testowanie małych kawałków naszego kodu
- piszemy kod (w pythonie), który testuje inny kod w pythonie
- skorzystamy z frameworka pytest

- czym będzie test jednostkowy?
    - będzie to zwykła funkcja, której nazwa będzie się zaczynała od “test_”, a która będzie coś testowała
    - taki zestaw funkcji, testów będziemy uruchamiali przy pomocy narzędzia pytest
    - https://martinfowler.com/bliki/GivenWhenThen.html
- podejście do pisania testów: TDD, Test Driven Development
- żeby w projekcie ustawić pytest, trzeba wejść do *Settings / Tools / Python Integrated Tools* i w sekcji *Testing* dla *Default test runner* wybrać *pytest*, później kliknąć w Fix na dole, aby zainstalować w projekcie pytest lub z konsoli (terminal) wydać polecenie *pip install pytest*

Argumenty

- na poziomie definicji funkcji
    - argumenty pozycyjne bez wartości domyślnej
    - argumenty pozycyjne z wartością domyślną
    - *args - będzie łapało wszystkie argumenty pozycyjne, których nie zdefiniowaliśmy bezpośrednio w funkcji, daje nam tuplę z wartościami przekazanymi jako argumenty pozycyjne w funkcji
    - **kwargs - będzie łapało wszystkie argumenty nazwane, których nie zdefiniowaliśmy bezpośrednio w funkcji, daje nam słownik, gdzie kluczem jest nazwa argumentu nazwanego a wartością, wartość tego argumentu
- na poziomie uruchomienia funkcji mamy argumenty:
    - pozycyjne (ang. positional arguments)
    - nazwane (ang. keyworded arguments)

OOP

OOP, Object Oriented Programming
To co do tej robiliśmy, to było programowanie funkcyjne.
Jak myślimy o programowaniu obiektowym to mamy dwie główne koncepcje:

- klasa - mówi jak ma być zbudowany obiekt
- obiekt - instancja klasy, zawiera strukturę zdefiniowaną w klasie i może już przechowywać konkretne wartości

Tworzenie nowej klasy:

- konwencja nazewnicza UpperCamelCase
- w jednym pliku możemy tworzyć wiele klas

Atrybuty (ang. attributes)

- “zmienne” w obiekcie, czyli pudełka, które trzymają wartości
- żeby każdy obiekt miał taki sam zestaw atrybutów, używamy metody `__init__`

Metody (ang. methods)

- czyli funkcje, które możemy uruchamiać na obiekcie
- w ramach metody mamy dostęp do atrybutów obiektu poprzez argument `self`
- funkcje definiujemy w klasie, czyli w schemacie, tak żeby każdy obiekt, który powstaje na podstawie klasy, miał ten sam zestaw metod

Właściwości (ang. property)
- są czymś pomiędzy atrybutem a metodą
- używane ze świata wyglądają jak zwykłe atrybuty
- definiowane w klasie są jako specjalnie oznaczone metody

Metody specjalne, tzw. dunder methods
- `__init__` - służy do ustawiania atrybutów w obiektach, za jej pomocą możemy powiedzieć, ustawić, jakie atrybuty każdy z obiektów ma mieć
- `__str__` - służy do obsługi tego jak python ma przerabiać konkretny obiekt na stringa
- https://www.python-course.eu/python3_magic_methods.php

Atrybuty klasowe (ang. class attributes)
- klasa w pythonie działa na kilka sposób, z jednej strony jest schematem do tworzenia obiektów, ale, możemy w niej trzymać również dane, które są wspólne dla wszystkich obiektów i które można wykorzystywać w całym programie -> do tego służą atrybuty klasowe
- jeżeli atrybut klasowy jest zapisany z dużych liter (np. `WORK_TIME`) to uznajemy, że nie powinien być modyfikowany, chociaż python nam tego nie zabroni

Metody statyczne
- siedzą w klasie
- nie są związane z żadnym obiektem
- najczęściej wykorzystujemy je do tworzenia metod narzędziowych, np. do przeliczania temperatur

Metody klasowe (ang. class methods)
- nie są związane z konkretnym obiektem
- są to metody, które pozwalają na tworzenie fabryk, czyli wygodniejszego sposobu tworzenia obiektów

Paradygmaty programowania obiektowego

https://pl.wikipedia.org/wiki/Programowanie_obiektowe
- Abstrakcja
  - aby określić, że jakaś klasa jest abstrakcyjna wykorzystujemy pythonowy moduł ABC (Abstract Base Class)
  - jeżeli mamy klasę oznaczoną jako abstrakcyjną, oznacza to, że ta klasa jest “schematem” dla klas, które po niej dziedziczą

    - może mieć metody, które są zaimplementowane i takie, których ciało trzeba zdefiniować w klasie dziedziczącej - takie metody nazywamy metodami abstrakcyjnymi
    - nie możemy tworzyć obiektów klas abstrakcyjnych (które dziedziczą po ABC i posiadają przynajmniej jedną metodę oznaczoną dekoratorem @abstractmethod)
- Hermetyzacja
    - https://pl.wikipedia.org/wiki/Hermetyzacja_(informatyka)
    - nazywana enkapsulacją albo kapsułkowaniem
    - ukrywanie implementacji
    - w kontekście metod
        - mamy kod zamknięty w ramach metody i spoza klasy nie modyfikuje tej metody
    - w kontekście atrybutów
        - w pythonie na poziomie języka nie ma jako takich modyfikatorów dostępu, definiujących skąd do jakiegoś atrybutu możemy sięgać, ale jest pewna konwencja
        - jak możemy nazywać atrybuty, żeby pokazać programiście, po które z nich może sięgać, a po które nie?
        - modyfikatory dostępu:
            - public (np. atrybut) - dostajemy się do atrybutu obiektu z każdego miejsca
            - protected (np. _atrybut) - to takie atrybuty, do których nie powinno się dostawać z zewnątrz, ale można się do nich dostawać z klas, które po tej klasie dziedziczą
            - private (np. __atrybut) - możemy dostawać się do tego atrybutu tylko z klasy, w której został zdefiniowany
- Dziedziczenie
    - klasy mogą po sobie dziedziczyć
    - dziedziczymy elementy z klasy rodzica i możemy je zmieniać, np. metody - mówimy wtedy o przykrywaniu metod
    - wielodziedziczenie
        - jedna klasa w pythonie może dziedziczyć po kilku klasach na raz
        - MRO - Method Resolution Order - który mówi, którą metodę python wybierze i dlaczego
        - https://docs.python.org/3/reference/datamodel.html#resolving-mro-entries
        - https://en.wikipedia.org/wiki/C3_linearization
- Polimorfizm
    - https://pl.wikipedia.org/wiki/Polimorfizm_(informatyka)
    - widać to szczególnie na przykładzie funkcji, gdzie w pythonie możemy używać `*args` i `**kwargs`

Relacja między klasami w programowaniu obiektowym

- agregacja i kompozycja - to relacje między klasami, takie, że jedna klasa zawiera w sobie obiekty drugiej klasy
    - agregacja - kiedy jeden obiekt może “sensownie” istnieć bez drugiego obiektu, np. Basket i Product, Zajęcia i Student, “zajęcia **zawierają** studentów”, strzałka zakończona pustym rombem
    - kompozycja - kiedy jeden obiekt NIE może sensownie istnieć bez drugiego obiektu, np. Dom i Pokój, “dom **posiada** pokoje”, strzałka zakończona pełnym rombem
- dziedziczenie
    - oznaczamy pustą strzałką

Wzorce projektowe

- https://pl.wikipedia.org/wiki/Wzorzec_projektowy_(informatyka)
- https://refactoring.guru/design-patterns/python

Wyjątki

- https://realpython.com/python-exceptions/
- https://docs.python.org/3/library/exceptions.html#exception-hierarchy

Moduły i pakiety

- https://realpython.com/python-modules-packages/
- moduły to pliki z kodem w Pythonie, pakiety, to foldery, które zawierają moduły
- po co robimy moduły i pakiety:
    - pojedynczy moduł powinien zawierać kod logicznie z nim związany, np. z obsługą raportów
    - kod zamknięty w moduły, możemy prosto wykorzystywać w wielu miejscach - unikamy redundancji kodu
    - łatwiejsze utrzymanie
- każdy plik z rozszerzeniem .py możemy potraktować jako moduł
- gdzie Python będzie szukał modułów i pakietów
    - będzie szukał w instalacji pythona podłączonej do projektu (czyli np. w condzie)
    - szuka w katalogach, które znajdują się w zmiennej środowiskowej PYTHONPATH
- skąd wziąć pakiety
    - z podstawowej instalacji pythona
    - https://pypi.org/

- dobrą praktyką jest umieszczanie wszystkich importów na samej górze pliku a nie gdzieś w środku
- kiedy importujemy plik to cały kod się uruchamia, w szczególności printy i inny kod który coś wykonuje a nie tylko definicje funkcji czy klas
- jeżeli uruchamiamy plik tools to zmienna __name__ jest w nim ustawiona na __main__, jeżeli importujemy w innym pliku, to jest ustawiona na nazwę modułu (czyli tools), jak to można wykorzystać?
    - możemy przygotować sobie plik, który będzie pełnił 2 role:
        - będzie zawierał narzędzia, które chcemy wykorzystywać w innych skryptach
        - możemy go uruchamiać i coś może wykonać
- w zależności od tego jak importujemy moduł, to co z niego przychodzi może nadpisać nasze zmienne, funkcje, klasy
    - `from moj_modul import suma`
        - jeżeli miałem wcześniej zdefiniowaną funkcję suma, to ta operacja spowoduje, że funkcja sprzed import zostanie zastąpiona tą z moj_modul
    - `from moj_modul import *`
        - zaimportuje wszystko i zastąpi elementy o tych samych nazwach, które miałem w programie, ale tylko takie, które były stworzone przed uruchomieniem instrukcji import

Operacje na plikach

- ladnie opisana lista trybów: https://stackoverflow.com/questions/16208206/confused-by-python-file-mode-w
    - `r` - czytanie (domyślny), `FileNotFoundError` jak nie ma pliku, wskaźnik na początku
    - `w` - pisanie, nadpisuje istniejący plik, tworzy nowy plik jak nie ma
    - `a` - dodawanie, wskaźnik na końcu pliku, tworzy nowy plik jak nie ma
    - `r+` - czytanie/pisanie, wskaźnik na początku, `FileNotFoundError` jak nie ma pliku
    - `w+` - czytanie/pisanie, nadpisuje istniejący plik, tworzy nowy plik jak nie ma
    - `a+` - czytanie/dodawanie, wskaźnik na końcu pliku, tworzy nowy plik jak nie ma
    - `rb, rb+, wb, wb+, ab, ab+` - analogicznie, ale w trybie binarnym
- standardowy sposób obsługi plików w pythonie to:
    - otwieramy plik
    - czytam/zapisujemy/wykonujemy operacje
    - zamykamy plik
- Parsowanie flag - https://www.programcreek.com/python/example/121/getopt.getopt
- moduł pythonowy os pozwala na rozszerzone operacja na plikach, jak zmianę nazwy, usuwanie, przenoszenie, itd.

Sposoby przechowywania/formatowania danych

- pliki
  - XML - https://en.wikipedia.org/wiki/XML
  - JSON - JavaScript Object Notation - to jest format, który bazuje na sposobie zapisywania obiektów w JavaScript, ale jest niezależny od tego języka, https://en.wikipedia.org/wiki/JSON , konstrukcja pliku JSON jest bardzo podobna do konstrukcji słowników w pythonie
  - YAML - https://pl.wikipedia.org/wiki/YAML
  - CSV, comma-separated values, poszczególne wartości rozdzielamy przecinkami, https://pl.wikipedia.org/wiki/CSV_(format_pliku)
  - INI - https://pl.wikipedia.org/wiki/INI

Zarządzanie bibliotekami w projekcie

- skąd brać biblioteki, które chcielibyśmy zainstalować w naszym projekcie?
  - https://pypi.org/
- jak znalezioną bibliotekę zainstalować?
  - `pip install nazwa_bilioteki`
- `pip freeze > requirements.txt`
- `pip install -r requirements.txt`

Data science

- jupyter notebook
  - jest to narzędzie, w którym tworzymy interaktywne dokumenty - połączenie zwykłego tekstu z kodem pythonowym
  - wersja online od google: https://colab.research.google.com/
  - markdown
    - https://medium.com/ibm-data-science-experience/markdown-for-jupyter-notebooks-cheatsheet-386c05aeebed
    - https://help.github.com/en/github/writing-on-github/basic-writing-and-formatting-syntax
- numpy
  - https://numpy.org/doc/1.17/
  - ndarray - n-dimensional array
  - cała fura operacji na tych tablicach
  - ćwiczenia z NumPy: https://github.com/rougier/numpy-100
- pandas
  - świetnie się sprawdza do danych tabelarycznych
  - wczytywanie danych / zapis danych do różnych formatów
  - serie
    - wartości - ndarray

    - indexy - mogą być różnych typów, nie tylko liczbowe (od 0 do …)
    - przechowuje dane homogeniczne
- DataFrame
    - jest to zestaw serii połączonych ze sobą, które dzielą ten sam indeks
    - każda seria ma jakąś nazwę