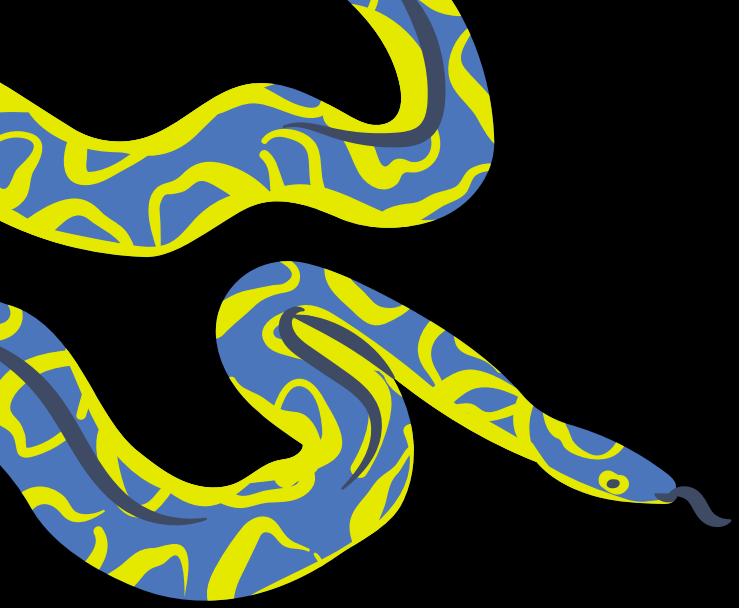
SciByteHub



Python

21.05

SciByteHub

Logowanie do komputerów

login: **nazimi1sbh** **(NAZwisko IMIe)**

hasło: **Edeibcabej024!**

SciByteHub

RECAP - Komendy linux

pwd	wyświetla bieżący katalog roboczy
ls	wyświetla zawartość katalogu
cd	zmienia bieżący katalog roboczy
mkdir	tworzy nowy katalog
rm -r	usuwa katalog razem z zawartością
touch	tworzy nowy pusty plik
rm	usuwa pliki
cp	kopiuje pliki
mv	przenosi lub zmienia nazwę
cat	wyświetla zawartość pliku
echo	wyświetla linię tekstu
grep	wyszukuje wzorce w tekście
man	wyświetla podręcznik użytkownika dla danej komendy

SciByteHub

RECAP - Komendy git

git init	inicjalizuje nowe repozytorium
git clone	kopiuje istniejące repozytorium
git pull	pobiera zmiany
git push	wysyła zmiany
git add *nazwa*	dodaje plik
git commit -a	zatwierdza wszystkie zmiany
git commit -m "*msg*"	zatwierdza zmiany z wiadomością bez edytora
git status	wyświetla stan repozytorium
git remote add nazwa url	dodaje nowe repozytorium

https://github.com/SciByteHub/Python-Kurs-ST-24

Typy zmiennych

SciByteHub

Typy zmiennych

string

np. “Hello World”, “123”

int

np. 23, 5, 0, 69420

float

np. 3.14, 1.23

bool

np. *True*, *False*

RECAP - struktury danych

SciByteHub

RECAP - struktury danych

list

lista elementów, które mogą mieć różne typy

np. [1, 2, 3], ['a', 'b', 'c'], [1, 'hello', True]

tuple

podobne do listy, ale elementy są niemodyfikowalne

np. (1, 2, 3), ('a', 'b', 'c')

dict

słownik, czyli zbiór par klucz-wartość

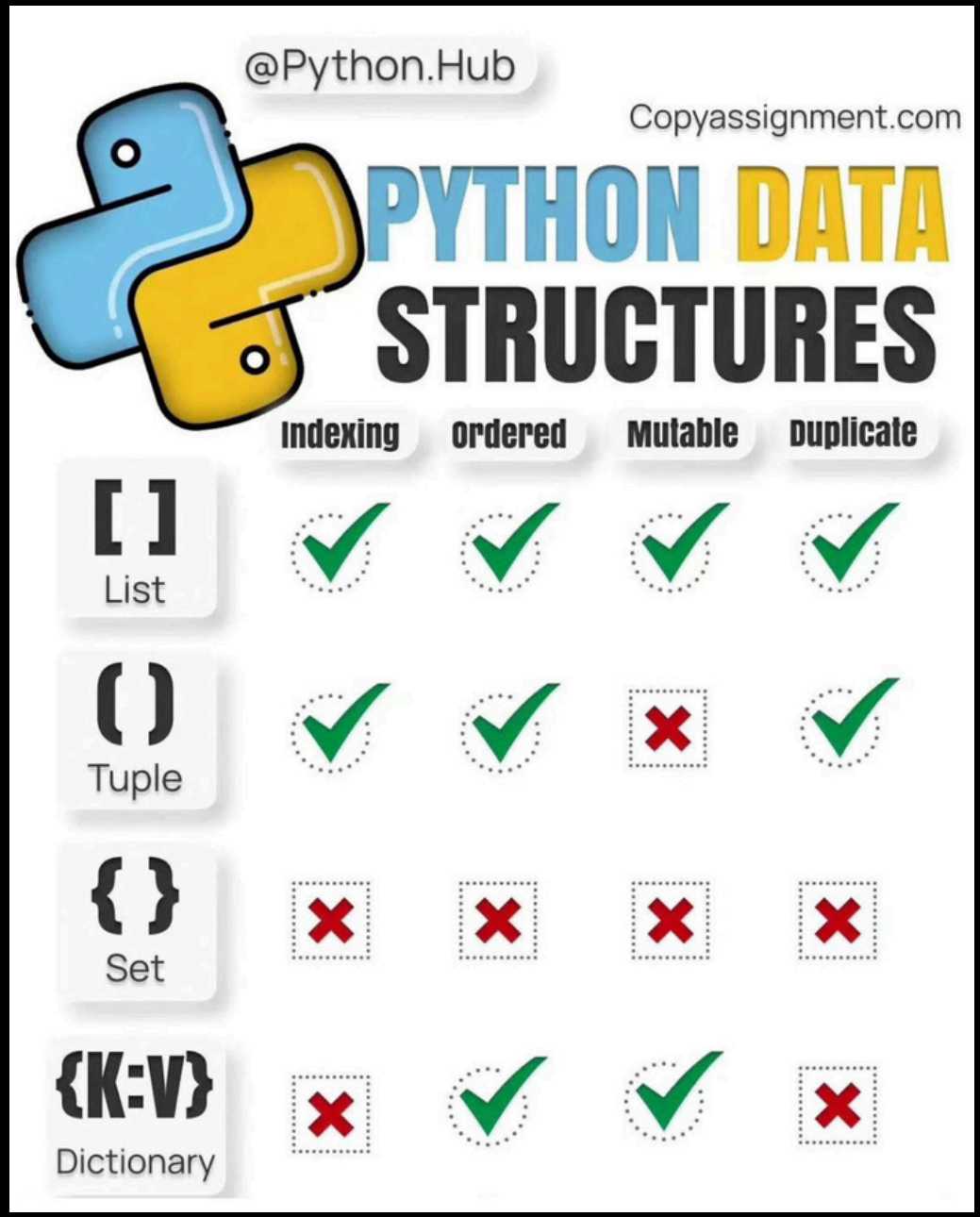
np. {'a': 1, 'b': 2, 'c': 3}

set

zbiór unikalnych elementów

np. {1, 2, 3}, {'a', 'b', 'c'}

SciByteHub


@Python.Hub
Copyassignment.com
PYTHON DATA STRUCTURES
Indexing
Ordered
Mutable
Duplicate
[ ]
List
( )
Tuple
{ }
Set
{K:V}
Dictionary

SciByteHub

RECAP - operacje na listach

indeksy	**pierwszy_element = lista[0]**
dodawanie elementów	**lista.append(6) lub lista.insert(2, 7)**
usuwanie elementów	**lista.remove(3), del lista[1],**
	usuniety_element = lista.pop()
slicing (wycinanie)	**podlista = lista[1:4]**
sortowanie	**lista.sort()**
znajdowanie indeksu elementu	**indeks = lista.index(4)**
długość listy	**dlugosc = len(lista)**
max i min	**maksimum = max(lista),**
	minimum = min(lista)
sprawdzanie obecności elementu	**czy_jest = 5 in lista**
czyszczenie listy	**lista.clear()**

RECAP - Pętle

SciByteHub

RECAP - Pętle

Pętla *for* (dla każdego) używana jest do wymieniania elementów z list, krotek, słowników, generatorów, itp.

Pętla *while* (dopóki) używana jest do tworzenia pętli nieskończonych lub wykonywania kodu w pętli pod jakimś warunkiem

RECAP - Jak działa for

```
lista = [”element1”, “element2”, “element3”]
for element in lista:
    print(element)
```

SciByteHub

RECAP - Jak działa while

```
while True:
    print("Pętla nieskończona")


i = 0
while i < 5:
    print("Warunek spełniony")
    i += 1
print("Pętla skończona. Tu wyjdziemy jak i będzie równe 5")
```

RECAP - funkcje

SciByteHub

RECAP - funkcje

```
def nazwa_funkcji(arg1, arg2):
    instrukcje
    . . .
    return output


output = nazwa_funkcji(a,b)
print(output)
```

RECAP - import

SciByteHub

RECAP - import

Importowanie bibliotek do kodu:

```
import biblioteka
```

Importowanie biblioteki jako alias:

```
import biblioteka as alias
```

SciByteHub

RECAP - Moduły i biblioteki

pip install SomePackage

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from statistics import mean
```

SciByteHub

RECAP - Generatory liczb pseudolosowych

Przykłady generatorów :

randint(0,10)	Zwraca liczbę naturalną z zadanego zakresu (np. od 0 do 10)
random()	Zwraca liczbę rzeczywistą między 0 a 1
uniform(0,10)	Zwraca liczbę rzeczywistą z zadanego zakresu (np. od 0 do 10)

SciByteHub

RECAP -Macierze i ich mnożenie

$$3 \cdot 0 + 1 \cdot 2 + 0 \cdot 0 = 2$$

$$\begin{bmatrix} 3 & 1 & 0 \\ 0 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 4 \\ 2 & 2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & \\ & \end{bmatrix}$$

2×3 $\quad$ 3×2 $\quad$ 2×2

SciByteHub

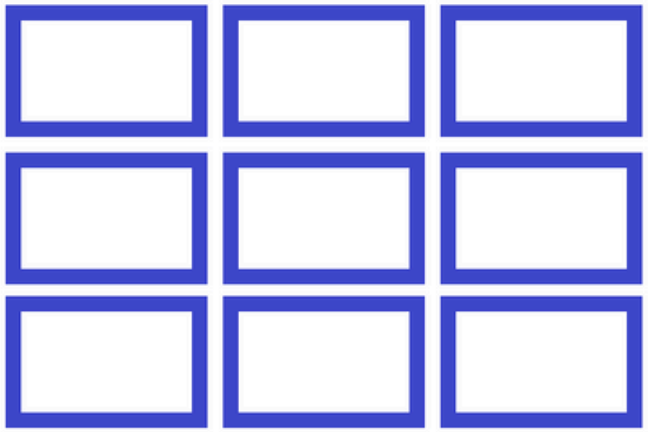
RECAP - Transponowanie macierzy

$$A = \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{2 \times 3} \qquad A^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}_{3 \times 2}$$

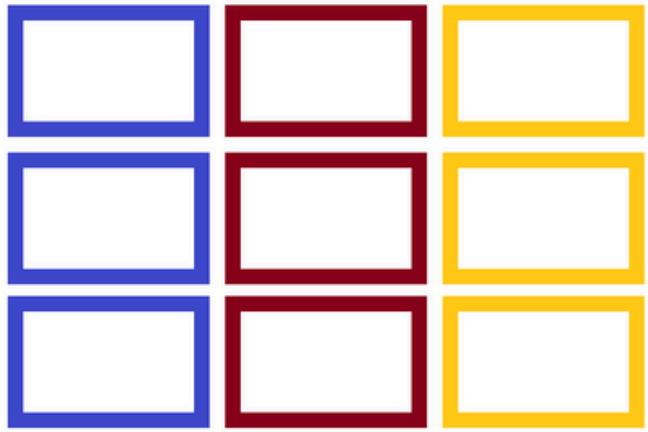
SciByteHub

RECAP - Macierz kontra Data Frame

Macierz



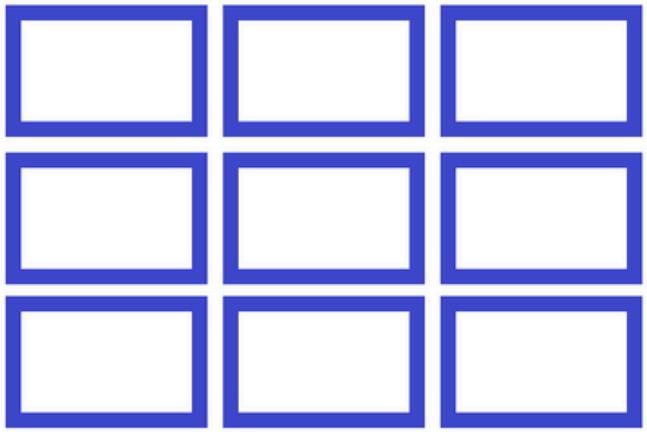
Data Frame



SciByteHub

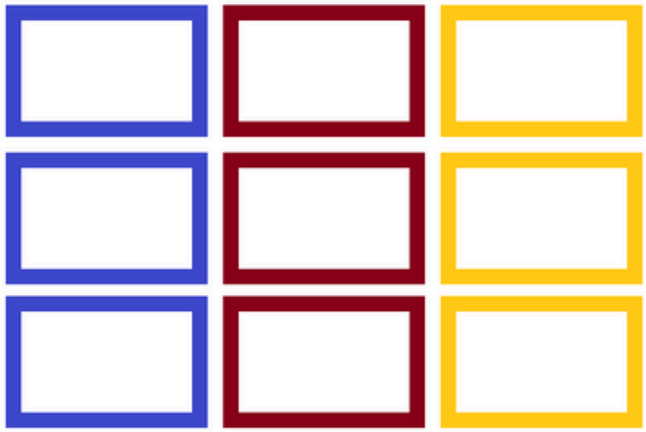
RECAP - Macierz kontra Data Frame

Macierz



Przechowuje tylko dane numeryczne

Data Frame



Przechowuje zarówno dane numeryczne, jak i stringi lub dane logiczne (boolean type - True, False)

SciByteHub

RECAP- try...except

```
try:
    x = input("Wprowadź liczbę całkowitą: ")
    x_num = int(x)
    print(f"Liczba Całkowita: {x_num}")
# Pobierze od urzytkownika tylko liczby całkowite, inaczej zwróci ValueError
except ValueError:
    print("Wprowadzona wartość nie jest liczbą całkowitą!")
    print("Program zostanie wyłączony!")
    exit(-1) # Wyłącza program z kodem -1(błąd, brak sukcesu, itp.)
```

SciByteHub

RECAP - Wywoływanie błędów ręcznie

Przykład:

```
x = 10
y = 0
if y == 0:
    raise ZeroDivisionError(“Nie można dzielić przez zero!”)
else:
    print(x/y)
```

RECAP - Punkt startu programu

SciByteHub

RECAP - Punkt startu programu

Punktem startowym programu nazywamy konstrukt:

```
if __name__ == "__main__":
    zrób coś :D
```

To spowoduje, że napisane pod tym konstruktem komendy zostaną wywołane tylko jeśli uruchomiony będzie plik macierzysty.

Importowanie skryptu nie spowoduje wykonania kodu spod tego konstruktu, bo parametr __name__ nie będzie "__main__".

SciByteHub

RECAP - klasy i obiekty

Klasa jest zbiorem parametrów i metod pod jedna wiadomą nazwą. Klasy mogą być odwzorowane wielokrotnie pod postacią obiektów.

Obiekt jest odwzorowaniem klasy. Każdy obiekt może różnić się parametrami od innych, nie zmieniając przy okazji parametrów początkowych klasy, z której powstał.

SciByteHub

RECAP - __init__

Jeśli chcemy wprowadzić konkretne wartości parametrów dla klasy przy definiowaniu obiektów musimy odpowiednio *zainicjować* klasę. Do tego używa się specjalnej metody *__init__*.

```
class MojaKlasa:
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2

obj1 = MojaKlasa("parametr1", "parametr2")
print(obj1.param1)
```

SciByteHub

RECAP - self

self odwołuje się bezpośrednio do nazwy obiektu lub klasy, tzn. jeśli mamy inicjację klasy za pomocą *__init__*, parametr *self* odwołuje się za nas do nazwy naszej klasy.

Inaczej: Jeśli nasza klasa ma nazwę MojaKlasa, parametr *self* będzie przyjmował nazwę MojaKlasa w różnych metodach klasy.

SciByteHub

RECAP - __str__

Metoda *__str__* pozwala nam na wypisanie czegoś przez *print* jeśli podamy tylko obiekt albo klasę bez definicji parametru który chcemy wyciągnąć.

```
class MojaKlasa:
    def __init__(self, param1, param2):
        self.param1 = param1
        self.param2 = param2
    def __str__(self):
        return f"Klasa MojaKlasa o parametrach {self.param1 = },   {self.param2 = }"

obj1 = MojaKlasa("parametr1", "parametr2")
print(obj1)
```

SciByteHub

RECAP - komenda assert

Przykład:

```
def divide(a, b):
    assert b != 0, "Error: division by zero"
    return a / b

result = divide(10, 2)
print("Result:", result)     # Output: Result: 5.0

result = divide(10, 0)       # AssertionError: Error: division by zero
```

Dziedziczenie

```python
# Definiujemy klase matke
class KlasaMatka:
    def __init__(self, parametr1, parametr2, parametr3):
        # Pozwól użytkownikowi wprowadzić 3 parametry oraz miej swoje 2 przypisane z góry
        self.parametr1 = parametr1
        self.parametr2 = parametr2
        self.parametr3 = parametr3
        self.parametr_matka1 = "Matka1"
        self.parametr_matka2 = "Matka2"

    # Jakieś metody
    def metoda1(self):
        print("Wykonywanie metody 1")

    def metoda2(self):
        print("Wykonywanie metody 2")

    # Skomplikowana funkcja do wypisywania informacji o klasie
    def __str__(self):
        # Pokaż nazwe klasy
        s1 = f"Klasa {self.__class__.__name__} o wprowadzonych parametrach"
        # Wygeneruj i pokaż spis parametrów klasy
        s2 = f"{', '.join([f'{_1}: {_2, type(_2).__name__}' for _1, _2 in vars(self).items()])}"
        # Połącz wygenerowane stringi w jedną całość z odstępem jednej spacji
        return ' '.join([s1, s2])
```

Dziedziczenie

```
obiekt1 = KlasaMatka( parametr1: 123,  parametr2: '456',  parametr3: [7, 8, 9])
print("Wykonywanie obiekt1.metoda1()")
obiekt1.metoda1()
print("Wykonywanie obiekt1.metoda2()")
obiekt1.metoda2()
print("Wypisywanie informacji o obiekcie klasy:")
print(obiekt1)
```

```
Wykonywanie obiekt1.metoda1()
Wykonywanie metody 1
Wykonywanie obiekt1.metoda2()
Wykonywanie metody 2
Wypisywanie informacji o obiekcie klasy:
Klasa KlasaMatka o wprowadzonych parametrach parametr1: (123, 'int'), parametr2: ('456', 'str'),
 parametr3: ([7, 8, 9], 'list'), parametr_wlasny: ('Coś', 'str'), parametr_wlasny2: ('Coś 2', 'str')

Process finished with exit code 0
```

Dziedziczenie

```python
# Klasa Dziecko dziedzicząca parametry z klasy KlasaMatka ale pobierający 4 zamiast 3 parametrów od użytkownika
class KlasaDziecko(KlasaMatka):
    def __init__(self, param1, param2, param3, param4):
        # super() inicjuje wszystkie parametry i metody z klasy macierzystej do klasy dziedziczącej.
        super().__init__(param1, param2, param3)
        self.parametr4 = param4
        self.parametr_dziecko = "Dziecko 1"
        self.parametr_dziecko2 = "Dziecko 2"
    # nadpisanie istniejącej w klasie macierzystej metody metoda2
    def metoda2(self):
        print("Nadpisana metoda 2 wykonana!")
```

Dziedziczenie

```
obiekt2 = KlasaDziecko( param1: 456,  param2: '789',  param3: [10, 11, 12],  param4: "Czwarty!")
print("Wykonywanie obiekt2.metoda1()")
obiekt2.metoda1()
print("Wykonywanie obiekt2.metoda2()")
obiekt2.metoda2()
print("Wypisywanie informacji o obiekcie klasy:")
print(obiekt2)
```

```
Wykonywanie obiekt2.metoda1()
Wykonywanie metody 1
Wykonywanie obiekt2.metoda2()
Nadpisana metoda 2 wykonana!
Wypisywanie informacji o obiekcie klasy:
Klasa KlasaDziecko o wprowadzonych parametrach parametr1: (456, 'int'), parametr2: ('789', 'str'), parametr3:
  ([10, 11, 12], 'list'), parametr_matka1: ('Matka1', 'str'), parametr_matka2: ('Matka2', 'str'), parametr4:
 ('Czwarty!', 'str'), parametr_dziecko: ('Dziecko 1', 'str'), parametr_dziecko2: ('Dziecko 2', 'str')

Process finished with exit code 0
```

SciByteHub