Python Wyjątki, klasy, biblioteki

Python Wyjątki, klasy, biblioteki

Agenda

1. **Programowanie obiektowe**
2. Wyjątki
3. Moduły
4. Biblioteki w Data Science

Python Wyjątki, klasy, biblioteki

Agenda

1. Programowanie obiektowe
2. **Wyjątki**
3. Moduły
4. Biblioteki w Data Science

Python Wyjątki, klasy, biblioteki

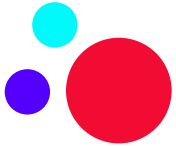
Agenda

1. Programowanie obiektowe
2. Wyjątki
3. **Moduły**
4. Biblioteki w Data Science

Python Wyjątki, klasy, biblioteki

Agenda

1. Programowanie obiektowe
2. Wyjątki
3. Moduły
4. **Biblioteki w Data Science**



Python Wyjątki, klasy, biblioteki

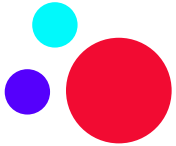
Programowanie obiektowe

Paradygmat programowania w którym programy składają się z obiektów:

- obiekty mają stany/właściwości (atrybuty, dane),
- zachowania (behaviors, metody).

Założenia programowania obiektowego:

- Abstrakcja
- Hermetyzacja
- Polimorfizm
- Dziedziczenie

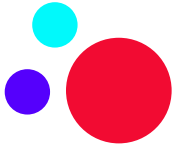


Python Wyjątki, klasy, biblioteki

Klasa

Klasa definiuje obiekt:

- jakie stany/właściwości może posiadać obiekt,
- jakie zachowania może posiadać obiekt (jakie metody są dostępne dla obiektu).



Python Wyjątki, klasy, biblioteki

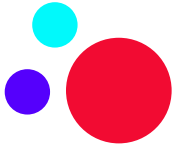
Klasa czy obiekt

Klasa

- Kot
- Budynek
- Samochód

Obiekt

- Filemon / Kot w butach
- Biały Dom / Wieża Eiffla
- BMW X5 / Fiat 126P



Python Wyjątki, klasy, biblioteki

Klasa czy obiekt

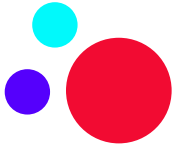
Instancja klasy kot:

Stany / właściwości

- imię = Filemon
- rasa = dachowiec
- stopień najedzenia = 0
- stopień wyspania = 10

Obiekt Filemon

- jedz()
- spij()



Python Wyjątki, klasy, biblioteki

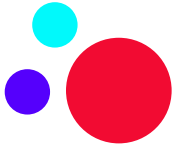
Klasa czy obiekt

Paradygmat programowania, w którym programy składają się z obiektów:

- obiekty mają stany/właściwości (atrybuty, dane),
- zachowania (behaviors, metody).

Klasa definiuje obiekt:

- jakie stany/właściwości może posiadać obiekt,
- jakie zachowania może posiadać obiekt (jakie metody są dostępne dla obiektu).

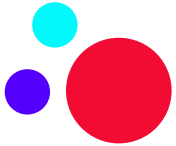


Python Wyjątki, klasy, biblioteki

Przykładowe klasy

DataFrame z pakietu pandas

```
import pandas as pd


df = pd.DataFrame({
    "x": [1, 2, 3],
    "y": [True, False, True]
})

df.shape  # atrybut

df.transpose()  # metoda
```

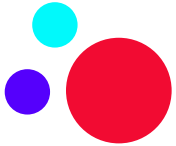


Python Wyjątki, klasy, biblioteki

Przykładowe klasy

Array z pakietu numpy

```
import numpy as np

array = np.array([[1, 2, 3], [True, False,
True]])

array.dtype  # atrybut

array.max()  # metoda
```



Python Wyjątki, klasy, biblioteki

Przykładowe klasy

Nasza przykładowa klasa:

```
class NazwaKlasy:

    zmienna = 0 # atrybut klasy

    def __init__(self, pole1, pole2): # inicjalizator (konstruktor)
        self.pole1 = pole1  # atrybut obiektu
        self.pole2 = pole2

    def nazwa_metody(self): #metoda, dzialanie na obiekcie
        "przykladowy kod metody"
        # tu miejsce na kod metody
```

Python Wyjątki, klasy, biblioteki

Dostęp do zmiennych w obiektach

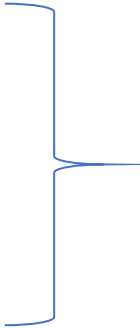
```
class Samochod:
    def __init__(self, marka, model):
        self.marka = marka  # atrybut obiektu
        self.model = model  # atrybut obiektu
        self.silnik_uruchomiony = False  # atrybut obiektu

    def uruchom_silnik(self):
        self.silnik_uruchomiony = True

# Tworzymy instancję klasy Samochod
moj_samochod = Samochod(marka="Toyota", model="Corolla")

# Dostęp do atrybutów obiektu
print(f"Marka samochodu: {moj_samochod.marka}")
print(f"Model samochodu: {moj_samochod.model}")
print(f"Stan silnika: {'Uruchomiony' if moj_samochod.silnik_uruchomiony else 'Zatrzymany'}")
```

Python Wyjątki, klasy, biblioteki

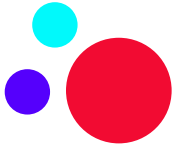
Nazywanie klas

Przykładowe dobrze nazwane klasy:

User
Invoice
OrderReceipt
NeuralNetwork



PascalCase

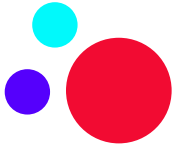


Python Wyjątki, klasy, biblioteki

Liczba parametrów funkcji __init__

```
class User:
    def __init__(self, name):
        self.name = name

user1 = User("Maciek")
user2 = User("Marta")
print(user1.name)  # Maciek
print(user2.name)  # Marta
```

parametry funkcji __init__

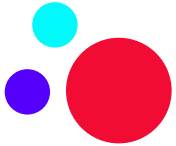


Python Wyjątki, klasy, biblioteki

Elementy prywatne


```
class BankAccount:
    # ...

    def _validate_user(self, token):
        # ...
        pass

    def make_transaction(self, token, transaction):
        self._validate_user(token)
        # ...
```

element prywatny

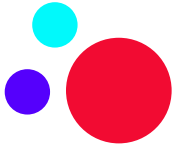


Python Wyjątki, klasy, biblioteki

Atrybuty klasy - SCREAMING_SNAKE_CASE

```
class TaxCalculator:
    VAT = 0.23
    # ...

class Direction:
    UP = 1
    DOWN = 2
    LEFT = 3
    RIGHT = 4

# Przykładowe użycie
direction = Direction.UP
if direction == Direction.UP:
    direction = Direction.DOWN
```

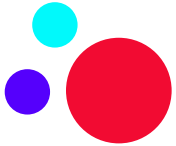
wartość, która nie ulega zmianie



Python Wyjątki, klasy, biblioteki

Atrybuty klasy - @staticmethod

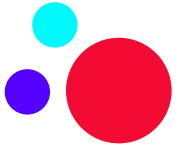
```
class Counter:
    num = 0

    def __init__(self):
        print("Tworzę")
        Counter.num += 1

    @staticmethod
    def print_counter():
        print(f"Stworzono {Counter.num}")

c1 = Counter()  # Tworzę
c2 = Counter()  # Tworzę
c3 = Counter()  # Tworzę
Counter.print_counter()  # Stworzono 3
```



Python Wyjątki, klasy, biblioteki

Przykładowe klasy

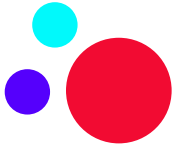
```
class Osoba:
    "Obiekt, ktory reprezentuje osobe oraz jej cechy"

    def __init__(self, imie, nazwisko, wiek, wzrost):
        "Konstruktor klasy osoba"
        self.imie = imie
        self.nazwisko = nazwisko
        self.wiek = wiek
        self.wzrost = wzrost

    def przedstaw_sie(self):
        "Przedstawiam dana osobe"
        print(f"Nazywam się {self.imie} {self.nazwisko} i mam {self.wiek}
lat")
```



Python Wyjątki, klasy, biblioteki

Przykładowe klasy

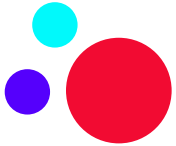
```
kamil = Osoba("Kamil", "Zet", 18, 180)

type(kamil)

kamil.przedstaw_sie()
```



Python Wyjątki, klasy, biblioteki

Przykładowe klasy

```
agatka = Osoba("Agata", "Igrek", 22, 160)
type(agatka)


agatka.przedstaw_sie()

agatka.imie

agatka.wiek = 24

agatka.__dict__


Osoba.__doc__
```



Python Wyjątki, klasy, biblioteki

Przykładowe klasy

Jeśli natrafimy na problem, pomocna może być funkcja help() - wyświetli nam atrybuty, opis oraz metody funkcji.

```
help(Osoba)
```

```
agatka.kobieta = True
```

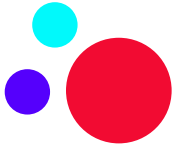
Wszystkie atrybuty znajdują się z zmiennej _dict_.

```
agatka.__dict__
```

```
kamil.__dict__
```

Zadanie 8.1 (instrukcja)

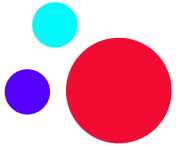
1. Stwórz klasę Kot o atrybutach 'imie', 'ilosc_przespanych_godzin' oraz metodzie: 'drzemka', która zwiększy atrybut ilość_przespanych_godzin o wartość podaną jako parameter do metody. Sprawdź działanie metody.



Python Wyjątki, klasy, biblioteki

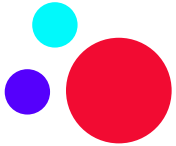
Dziedziczenie

Koncept programowania obiektowego pozwalający na definiowanie nowych klas w oparciu o istniejące, rozszerzając lub zmieniając ich funkcjonalność.



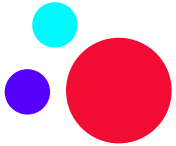
Python Wyjątki, klasy, biblioteki
Dziedziczenie

```
class Osoba:
    "Obiekt, ktory reprezentuje osobe oraz jej cechy"

    def __init__(self, imie, nazwisko, wiek, wzrost):
        "Konstruktor klasy osoba"
        self.imie = imie
        self.nazwisko = nazwisko
        self.wiek = wiek
        self.wzrost = wzrost

    def przedstaw_sie(self):
        "Przedstawiam dana osobe"
        print(f"Nazywam się {self.imie} {self.nazwisko} i mam {self.wiek}
lat")
```



Python Wyjątki, klasy, biblioteki
Dziedziczenie

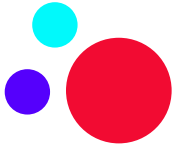
```
class Kursant(Osoba): #dziedziczy po klasie Osoba

    def __init__(self, imie, nazwisko, wiek, wzrost, ocena):
        super().__init__(imie, nazwisko, wiek, wzrost)
        self.ocena = ocena

    def czy_obecny_na_zajeciach(self):
        print("Przykro mi, mamy awarię systemu, nie mam takiej informacji.")
```



Python Wyjątki, klasy, biblioteki

Dziedziczenie

```
asia = Kursant("Asia", "Iks", 24, 160, 70)

type(asia)

help(asia)

asia.przedstaw_sie()

asia.czy_obecny_na_zajeciach()


kasia = Osoba("Asia", "Iks", 24, 160)
isinstance(kasia, Osoba)
```



Python Wyjątki, klasy, biblioteki

Dziedziczenie

isinstance(asia, Osoba)

isinstance(kasia, Kursant)

isinstance(asia, Kursant)

asia.__class__

asia.__class__.__name__

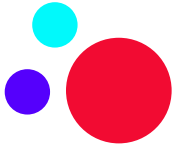
Zadanie 8.2 (instrukcja)

1. Stwórz hierarchię klas reprezentujących różne rodzaje pojazdów w systemie zarządzania pojazdami. Bazową klasą powinna być klasa Pojazd, a następnie utwórz co najmniej dwie pochodne klasy, np. Samochód i Motocykl. Każda z tych klas powinna posiadać unikalne cechy oraz metody charakterystyczne dla danego rodzaju pojazdu.

Python Wyjątki, klasy, biblioteki
Polimorfizm

```
class Zwierzę:
    def dźwięk(self):
        pass

class Kot(Zwierzę):
    def dźwięk(self):
        return "Miau!"

class Pies(Zwierzę):
    def dźwięk(self):
        return "Hau!"

zwierzęta = [Kot(), Pies()]

for zwierzę in zwierzęta:
    print(zwierzę.dźwięk())
```

Użycie polimorfizmu

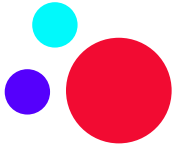


Python Wyjątki, klasy, biblioteki

Przeciążanie metod

Przeciążanie metod to bardzo istotny element programowania obiektowego. Umożliwia budowanie abstrakcji oraz polimorfizmu.

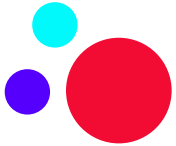
Interpreter Python'a wie, jakiej klasy jest obiekt i wywołuje odpowiednią metodę.



Python Wyjątki, klasy, biblioteki

Przeciążanie metod

```
class DSKursant(Kursant): #dziedziczy po klasie Kursant
    def __init__(self, imie, nazwisko, wiek, wzrost, ocena):
        super().__init__(imie, nazwisko, wiek, wzrost, ocena)

    def czy_obecny_na_zajeciach(self):
        print("Oczywiscie, kursanci nie opuszczają zajęć!"
```



Python Wyjątki, klasy, biblioteki

Przeciążanie metod

```
basia = DSKursant("Barbara", "Wu", 30, 160, 90)

help(basia)

basia.przedstaw_sie()

basia.czy_obecny_na_zajeciach()

asia.czy_obecny_na_zajeciach()
```

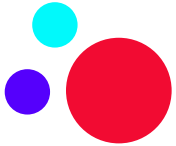
Zadanie 8.3

Metody magiczne (instrukcja)

Stwórz klasy Kot i dziedziczącą po niej klasę GrubyKot. Atrybuty klasy Kot to 'imie', 'ilosc_przespanych_godzin' Atrybuty klasy GrubyKot to 'imie', 'ilosc_przespanych_godzin' i 'ilość_zjedzonych_lasagne'.

Zaimplementuj dwie metody: - zamiaucz - wypisującą string: " mówi miauuu" - pobiegnij: (Kot) wypisującą: "skoro muszę..." jeśli 'ilosc_przespanych_godzin' jest mniejsza niż 10, w przeciwnymrazie wypisać "już lecę" (GrubyKot) wypisującą: "no doobra" jeśli 'ilość_zjedzonych_lasagne' jest mniejsza niż 3, w przeciwnym razie wypisać "chyba śnisz"

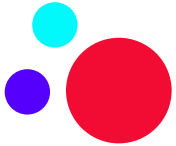
Stwórz obiekty: Filemon (Kot) i Garfield (GrubyKot) i upewnij się, że wszystkie atrybut i metody są poprawnie zaimplementowane.



Python Wyjątki, klasy, biblioteki

Dziedziczenie

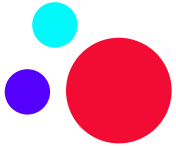
```
class Osoba:
    "Obiekt, ktory reprezentuje osobe oraz jej cechy"

    def __init__(self, imie, nazwisko, wiek, wzrost):
        "Konstruktor klasy osoba"
        self.imie = imie
        self.nazwisko = nazwisko
        self.wiek = wiek
        self.wzrost = wzrost

    def przedstaw_sie(self):
        "Przedstawiam dana osobe"
        print(f"Nazywam się {self.imie} {self.nazwisko} i mam {self.wiek} lat")
```



Python Wyjątki, klasy, biblioteki

Atrybuty

```
asia = Osoba("Asia", "Kowalska", 30, 160)

getattr(asia, "imie")

hasattr(asia, "nazwisko")

hasattr(asia, "kobieta")

setattr(asia, "imie", "Joanna")
```

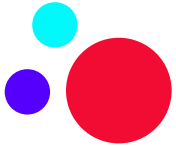


Python Wyjątki, klasy, biblioteki

Metody magiczne

Metody magiczne to funkcje, które należą do klasy. Mogą to być zarówno instancje, jak i metody klasy. Można je łatwo zidentyfikować, ponieważ wszystkie zaczynają się i kończą podwójnym podkreśleniem:

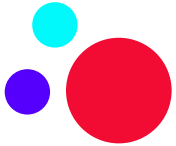
__actual_name__ ← **dunder** – double underscores



Python Wyjątki, klasy, biblioteki

Metody magiczne - __init__

```
class Osoba:
    def __init__(self, imie, nazwisko):
        self.imie = imie
        self.nazwisko = nazwisko
```



Python Wyjątki, klasy, biblioteki

Metody magiczne - __iter__

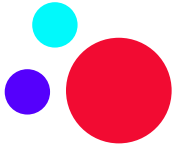
```
class LiczbyParzyste:
    def __init__(self, limit):
        self.limit = limit

    def __iter__(self):
        self.liczba = 0
        return self

    def __next__(self):
        if self.liczba < self.limit:
            wynik = self.liczba
            self.liczba += 2
            return wynik
        else:
            raise StopIteration
```

```
liczby = LiczbyParzyste(10)

for liczba in liczby:
    print(liczba)
```

Python Wyjątki, klasy, biblioteki

Metody magiczne - __len__

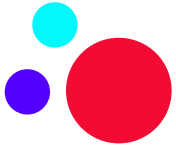
```
class Kolekcja:
    def __init__(self, elementy):
        self.elementy = elementy

    def __len__(self):
        return len(self.elementy)
```



Python Wyjątki, klasy, biblioteki

Metody magiczne - __contains__

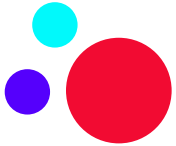
```
class PrzykladowaKlasa:
    def __init__(self, elementy):
        self.elementy = elementy

    def __contains__(self, szukany_element):
        return szukany_element in self.elementy

# Użycie
moja_klasa = PrzykladowaKlasa([1, 2, 3, 4, 5])

# Sprawdzenie, czy element jest zawarty
czy_jest = 3 in moja_klasa
print(czy_jest)  # True

czy_nie_ma = 6 in moja_klasa
print(czy_nie_ma)  # False
```



Python Wyjątki, klasy, biblioteki

Metody magiczne - __getitem__

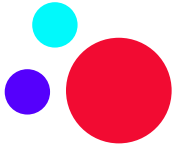
```
class PrzykladowaKlasa:
    def __init__(self, elementy):
        self.elementy = elementy

    def __getitem__(self, indeks):
        return self.elementy[indeks]

# Użycie
moja_klasa = PrzykladowaKlasa([1, 2, 3, 4, 5])

# Pobranie elementu za pomocą __getitem__
pierwszy_element = moja_klasa[0]
print(pierwszy_element)  # 1

# Można również używać w pętli for
for element in moja_klasa:
    print(element)
```



Python Wyjątki, klasy, biblioteki

Metody magiczne - __str__

```
class PrzykladowaKlasa:
    def __init__(self, imie, wiek):
        self.imie = imie
        self.wiek = wiek

    def __str__(self):
        return f"{self.imie}, {self.wiek} lat"

# Użycie
osoba = PrzykladowaKlasa("Anna", 25)

# Wywołanie metody magicznej __str__
repr_osoba = str(osoba)
print(repr_osoba)  # "Anna, 25 lat"
```



Python Wyjątki, klasy, biblioteki

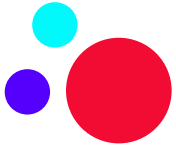
Metody magiczne

```
class Osoba:
    "Obiekt, ktory reprezentuje osobe oraz jej cechy"

    def __init__(self, imie, nazwisko, wiek, wzrost):
        "Konstruktor klasy osoba"
        self.imie = imie
        self.nazwisko = nazwisko
        self.wiek = wiek
        self.wzrost = wzrost

    def przedstaw_sie(self):
        "Przedstawiam dana osobe"
        print(f"Nazywam się {self.imie} {self.nazwisko} i mam {self.wiek} lat")
```

Python Wyjątki, klasy, biblioteki

Metody magiczne

```
class Osoba:
    "Obiekt, ktory reprezentuje osobe oraz jej cechy"

    def __init__(self, imie, nazwisko, wiek, wzrost):
        "Konstruktor klasy osoba"
        self.imie = imie
        self.nazwisko = nazwisko
        self.wiek = wiek
        self.wzrost = wzrost

    def przedstaw_sie(self):
        "Przedstawiam dana osobe"
        print(f"Nazywam się {self.imie} {self.nazwisko} i mam {self.wiek} lat")

    def __str__(self):
        return f"{self.imie} {self.nazwisko}"
```



Python Wyjątki, klasy, biblioteki

Metody magiczne


```
asia = Osoba("Asia", "Kowalska", 30, 160)
```

Zwraca czytelną reprezentację tekstową obiektu:

```
print(asia)
```

Asia Kowalska, wiek: 30, wzrost: 160 cm

```
str(asia)
```

'Asia Kowalska, wiek: 30, wzrost: 160 cm'

Python Wyjątki, klasy, biblioteki

Metody magiczne - podsumowanie

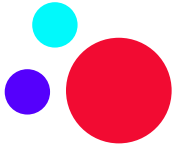
infoShareAcademy.com

Zadanie 8.4

Metody magiczne (instrukcja)

1. Zaktualizuj klasę Osoba o metodę magiczną __eq__, która pozwala na sprawdzenie równości dwóch obiektów.

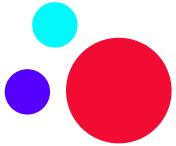
```
class Osoba:
    "Obiekt, ktory reprezentuje osobe oraz jej cechy"

    def __init__(self, imie, nazwisko, wiek, wzrost):
        "Konstruktor klasy osoba"
        self.imie = imie
        self.nazwisko = nazwisko
        self.wiek = wiek
        self.wzrost = wzrost

    def przedstaw_sie(self):
        "Przedstawiam dana osobe"
        print(f"Nazywam się {self.imie} {self.nazwisko} i mam {self.wiek} lat")
```



Python Wyjątki, klasy, biblioteki

Zasady SOLID

- Single Responsibility Principle [SRP]
- Open Closed Principle [OCP]
- Liskov Substitution Principle [LSP]
- Interface Segregation Principle [ISP]
- Dependency Inversion Principle [DIP]

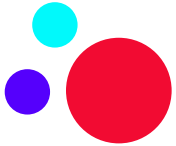


Python Wyjątki, klasy, biblioteki

Single Responsibility Principle [SRP]

```
class Employee():
    def calculate_payment(self):
        ...

    def report_hours(self):
        ...

    def save(self):
        ...
```

```
class EmployeePaymentCalculator():
    def calculate_payment(self, employee):
        ...

class EmployeeTimeReport():
    def report_hours(self, employee):
        ...

class EmployeeRepository():
    def save(self, employee):
        ...
```

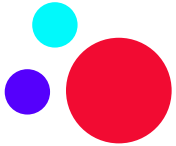


Python Wyjątki, klasy, biblioteki

Open Closed Principle [OCP]

```
class Profession(Enum):
    ARTIST = auto()
    LAWYER = auto()
    TEACHER = auto()

@dataclass
class TaxPayer:
    profession: str
    salary: float

def calculate_tax(tax_payer):
    if tax_payer.profession == Profession.ARTIST:
        return tax_payer.salary * 0.25
    elif tax_payer.profession == Profession.LAWYER:
        return tax_payer.salary * 0.34
    elif tax_payer.profession == Profession.TEACHER:
        return tax_payer.salary * 0.15
```

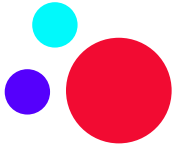
```
@dataclass
class TaxPayer(ABC):
    salary: float

    @abstractmethod
    def calculate_tax(self):
        pass

class Artist(TaxPayer):
    profession = Profession.ARTIST

    def calculate_tax(self):
        return self.salary * 0.25

class Lawyer(TaxPayer):
    profession = Profession.LAWYER

    def calculate_tax(self):
        return self.salary * 0.34

class Teacher(TaxPayer):
    profession = Profession.TEACHER

    def calculate_tax(self):
        return self.salary * 0.15

artist = Artist(1000)
artist.calculate_tax()  # 250.0
```



Python Wyjątki, klasy, biblioteki

Liskov Substitution Principle [LSP]

```
class ReadableFile(ABC):
    @abstractmethod
    def read(self) -> str: ...

class WritableFile(ABC):
    @abstractmethod
    def write(self, input_text: str) -> None: ...

class NormalFile(ReadableFile, WritableFile):
    def read(self) -> str:
        # read file implementation
        print('Reading from file')

    def write(self, input_text: str) -> None:
        # write file implementation
        print('Writing to file')

 class ReadonlyFile(ReadableFile):
    def read(self) -> str:
        # read readonly file implementation
        print('Reading from readonly file')
```



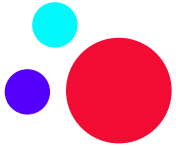
Python Wyjątki, klasy, biblioteki

Interface Segregation Principle [ISP]

```
# Zły design z naruszeniem ISP
class Worker(ABC):
    @abstractmethod
    def work(self):
        pass

    @abstractmethod
    def eat(self):
        pass
```

```
# Poprawiony design, interfejsy są segregowane
class Workable(ABC):
    @abstractmethod
    def work(self):
        pass

class Eatable(ABC):
    @abstractmethod
    def eat(self):
        pass
```

```
# Klasa konkretna implementująca Worker, ale nie używająca metody eat
class Programmer(Worker):
    def work(self):
        print("Coding...")

    def eat(self):
        pass  # Ta metoda nie jest używana przez Programistę
```



Python Wyjątki, klasy, biblioteki

Dependency Inversion Principle [DIP]

```
class Task:
    def process(self) -> None:
        # some processing things...
        email_sender = EmailSender()
        email_sender.send('some nice message')
```

```
class EmailSender:
    def send(self, message: str) -> None:
        # send email
        print(f'Sending email with message: {message}')
```

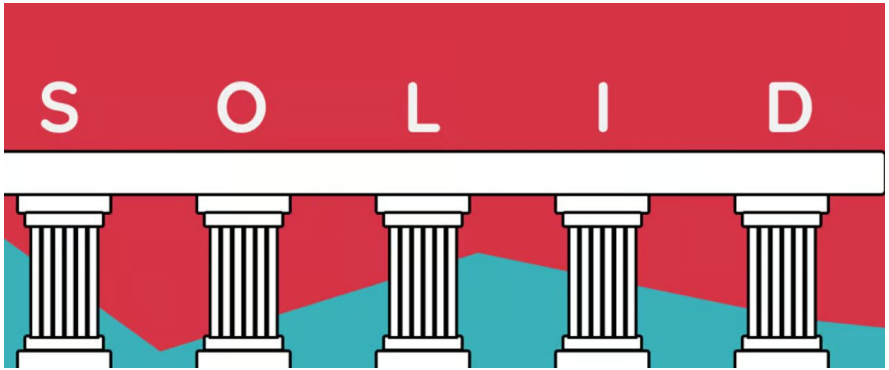
```
class MessageSender(ABC):
    @abstractmethod
    def send(self, message: str) -> None: ...
```

```
class EmailSender(MessageSender):
    def send(self, message: str) -> None:
        # send email
        print(f'Sending email with message: {message}')
```

```
class Task:
    def __init__(self, message_sender: MessageSender):
        self.message_sender = message_sender

    def process(self) -> None:
        # some processing things...
        self.message_sender.send('some nice message')
```

Python Wyjątki, klasy, biblioteki

SOLID



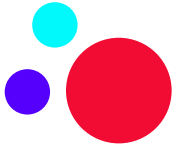
Zadanie 8.5

Podsumowanie (instrukcja)

1. Stwórz prosty system do obsługi pracowników w firmie. Każdy pracownik ma swoje podstawowe informacje, takie jak imię, nazwisko, pensja. System powinien umożliwiać obliczanie premii dla pracowników oraz generowanie raportu z informacjami o pracownikach.

Python Wyjątki, klasy, biblioteki

Programowanie obiektowe - podsumowanie

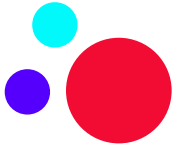


Python Wyjątki, klasy, biblioteki

Wyjątki

Wyjątek to mechanizm do obsługi niespodziewanych zdarzeń. Generowany jest, gdy pojawi się sytuacja nieprzewidziane oraz nie obsłużona przez programistę.

Język Python ma zbiór wyjątków, które są przez niego zwracane w sytuacjach, które w świetle kodu nie powinny mieć miejsca.



Python Wyjątki, klasy, biblioteki

Rodzaje błędów

1. Błąd nazwy (Name error)
2. Błąd typu (Type error)
3. Błąd wartości (Value error)
4. Błąd składniowy (Syntax error)
5. Błąd wcięcia (Indentation error)

Python Wyjątki, klasy, biblioteki

Błędy składni (SyntaxError, IndentationError)

Zgłaszane przed uruchomieniem programu przykładowe błędy:

- brak dwukropka,
- brak zamknięcia nawiasu,
- złe wcięcia.

```
for i in [1, 2]
  print(i)
```

brak dwukropka

```
Input In [3]
  for i in [1, 2]
                  ^
SyntaxError: expected ':'
```

```
def funkcja_przywitaj():
print("Witaj!")
```

brak dwukropka

```
Input In [5]
  print("Witaj!")
  ^
IndentationError: expected an indented block after function definition on line 1
```

Python Wyjątki, klasy, biblioteki

Wyjątki

Błędy wykryte w trakcie wykonywania naszego programu:

np. dzielenie przez 0,

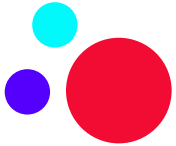
1 / 0

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
c:\Users\Lenovo\OneDrive\Pulpit\InfoShare\testowy\test.ipynb Cell 5 line <cell line: 1>()
----> 1 1 / 0

ZeroDivisionError: division by zero
```

czy wyjście poza dostępny zakres.

a = [1, 2]
a[2]

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
c:\Users\Lenovo\OneDrive\Pulpit\InfoShare\testowy\test.ipynb Cell 5 line <cell line: 2>()
      1 a = [1, 2]
----> 2 a[2]

IndexError: list index out of range
```



Python Wyjątki, klasy, biblioteki

Wbudowane wyjątki

- ZeroDivisonError - dzielę przez 0
- AttributeError - błędne odwołanie do atrybutu obiektu
- ImportError - problem z importem modułu
- KeyboardInterrupt - program przerwany przez Ctrl+c
- IndexError - brak indexu, np. w liście
- KeyError - brak klucza w słowniku
- NameError - brak zmiennej o podanej nazwie
- IOError - np. problem z otworzeniem pliku
- SyntaxError - błąd składni
- IndentationError - nieprawidłowe wcięcie
- TypeError - zły typ danych, np. str zamiast int
- ValueError - nieprawidłowa wartość

Python Wyjątki, klasy, biblioteki

Podnoszenie wyjątku (raise exception)

raise NazwaBledu('argument wyjatku')

raise Exception('Wystapil zdefiniowany przeze mnie blad')

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
c:\Users\Lenovo\OneDrive\Pulpit\InfoShare\testowy\test.ipynb Cell 5 line <cell line: 1>()
----> 1 raise Exception('Wystapil zdefiniowany przeze mnie blad')

Exception: Wystapil zdefiniowany przeze mnie blad
```

raise ZeroDivisionError('Wyswietlam blad klasy ZeroDivisionError')

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
c:\Users\Lenovo\OneDrive\Pulpit\InfoShare\testowy\test.ipynb Cell 5 line <cell line: 1>()
----> 1 raise ZeroDivisionError('Wyswietlam blad klasy ZeroDivisionError')

ZeroDivisionError: Wyswietlam blad klasy ZeroDivisionError
```

Zadanie 8.6

raise exception (instrukcja)

1. Stwórz funkcję "bmi", która przyjmuje za parametry wagę i wzrost. Jeśli którykolwiek z parametrów jest mniejszy lub równy 0, zwróć wyjątek Exception, który wypisze wiadomość: "Kłamczuszek".

Python Wyjątki, klasy, biblioteki

Podstawowa obsługa wyjątków przy użyciu try i except

try:
  "kod do wypróbowania"
except:
  "kod w przypadku błędu"

```
try:
    foo = int(input("podaj liczbe: "))
    print("Wpisales ", foo)
except:
    print("Nie wpisales liczby")
```

Zadanie 8.7

try except (instrukcja)

1. Stwórz funkcję "bmi_obsluzony_blad", która wywoła funkcję "bmi". Jeśli funckja "bmi" zwróci jakikolwiek błąd należy go złapać i wypisać komunikat "Wpisałeś niepoprawne dane".

Python Wyjątki, klasy, biblioteki

Przechwytywanie argumentu funkcji

```
try:
    1 / 0
except Exception as e:
    print(f"Opis błędu: '{e}'")
```

nie złapie BaseException, SystemExit, KeyboardInterrupt GeneratorExit..

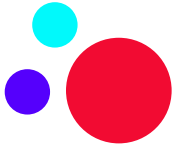
Opis błędu: ‘ division by zero ‘

Python Wyjątki, klasy, biblioteki

Zdefiniowany typ wyjątku

```
try:
    "kod do wypróbowania"
except JakisError:
    "kod w przypadku błędu"
```

bezpośrednie zdefiniowanie wyjątku

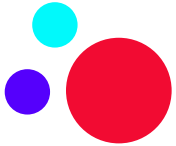


Python Wyjątki, klasy, biblioteki

Zdefiniowany typ wyjątku

```
try:
  print(bmi(74, 182))
    2/0
    print("Test drukowania")
except ZeroDivisionError:
    print('Nie dziel przez 0 ty ...')
except Exception as e:
    print(f"Nieznany błąd - skontaktuj się z twórcą kodu! Opis błędu {e}")
except IndexError:
    print("Prawdopodobnie podałeś złe dane")
```

tu otrzymamy wyjątek klasy ZeroDivisionError

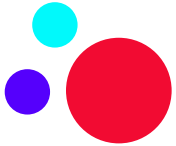


Python Wyjątki, klasy, biblioteki

Zdefiniowany typ wyjątku


```
import sys

try:
    foo = int(input("podaj liczbe: "))
    print("Wpisales ", foo)
except:
    print("Nieoczekiwany blad", sys.exc_info()[0])
```

złapie wszystko



Python Wyjątki, klasy, biblioteki

try except else

try:

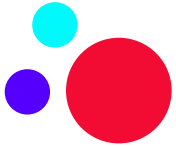
"kod do wypróbowania"

except JakisError:

"kod w przypadku błędu"

else:

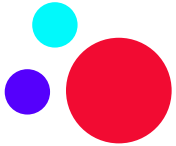
"kod w przypadku braku błędu"



Python Wyjątki, klasy, biblioteki

try except else

```
try:
    foo = int(input("podaj liczbe: "))
except ValueError:
    print("Nie wpisales liczby")
else:
    print("Wpisales ", foo)
```



Python Wyjątki, klasy, biblioteki

try except else finally

try:

"kod do wypróbowania - potencjalnie niebezpieczny"

except:

"kod w przypadku błędu - kod obsługi wyjątku"

else:

"kod w przypadku braku błędu"

finally:

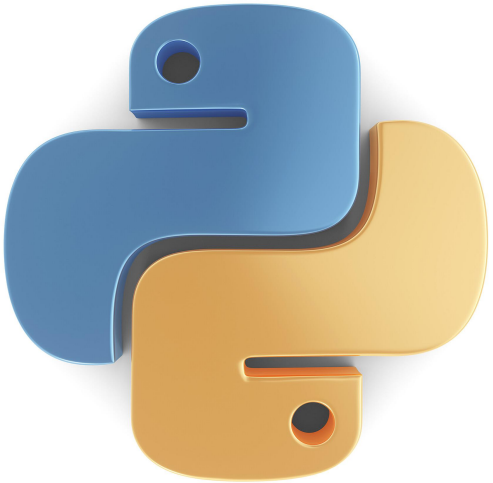
"kod który wykona się zawsze - akcje kończące blok"

Python Wyjątki, klasy, biblioteki
finally

```
try:
    foo = int(input("podaj liczbe: "))
except ValueError:
    print("Nie wpisales liczby")
else:
    print("Wpisales ", foo)
finally:
    print("To wykona się zawsze na koniec niezaleznie, czy wystąpił wyjątek, czy też nie")
```

Python Wyjątki, klasy, biblioteki

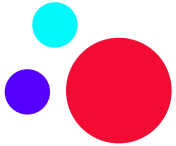
Wyjątki - podsumowanie



infoShareAcademy.com

Zadanie 8.8 (instrukcja)

1. Zmodyfikuj funkcję z poprzedniego zadania (zwracającą bmi) w taki sposób, żeby printowala zawsze informację o sumie wagi i wzrostu, a jeśli nie wystąpił błąd to wypisała 'BMI prawidłowe' jeśli wartość jest mniejsza od 25, w przeciwnym razie 'Skontaktuj się ze specjalistą'.



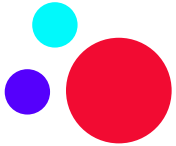
Python Wyjątki, klasy, biblioteki

Moduły, pakiety, biblioteki

Biblioteka to kolekcja pakietów.

Pakiet to kolekcja modułów (zbiór funkcji, nowych typów danych i innych obiektów, które rozszerzają możliwości środowiska Python).

Moduł to plik z kodem źródłowym w Pythonie z rozszerzeniem .py (każdy skrypt może być modułem).



Python Wyjątki, klasy, biblioteki

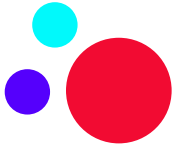
Moduły, pakiety, biblioteki – Anaconda vs Python

Anaconda

- anaconda-navigator
- preinstalowane moduły
- oprócz Pythona zawiera R, Jupyter Notebook...
- mniej dostępnych pakietów
- conda do instalacji pakietów

Python

- tylko język programowania
- dużo więcej dostępnych pakietów
- pip do instalacji pakietów
- szybciej dostępne nowe wersje



Python Wyjątki, klasy, biblioteki
Pakiety

Zarządzanie pakietami:

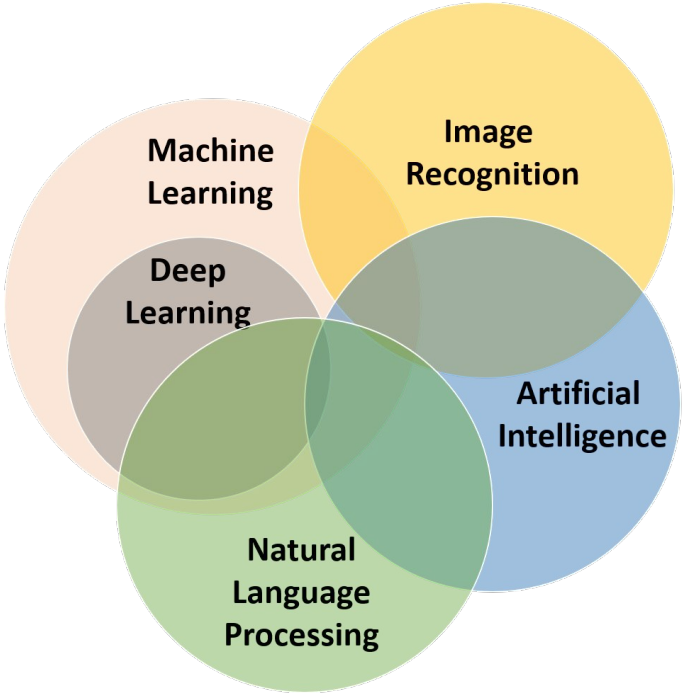
- Conda (korzystamy jeśli używamy z Anacondy/Minicondy)
- PIP (używamy kiedy korzystamy z Python'a lub kiedy conda nie posiada pakietu)

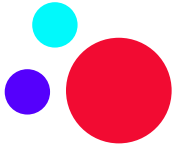
Przykładowe pakiety:

- NumPy
- Pandas
- Matplotlib
- Seaborn
- Plotly
- OpenCV
- Scikit-learn

Python Wyjątki, klasy, biblioteki

Interakcje między pakietami

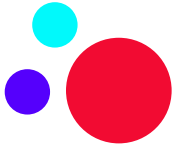




Python Wyjątki, klasy, biblioteki
PIP

PIP - system zarządzania pakietami używany do instalacji i zarządzania pakietami, które można znaleźć w Indeks pakietów Pythona (PyPI): https://pypi.org/

```
!conda --version

conda install seaborn  # Nie instalujemy z jupytera! Instalujemy z command line

import seaborn

import this

!conda install scipy=1.6.2

!pip install -r requirements.txt
```

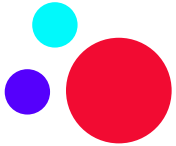


Python Wyjątki, klasy, biblioteki

Korzystanie z bibliotek

Import biblioteki:

```
import nazwa_biblioteki
nazwa_biblioteki.funkcja_z_biblioteki()

import nazwa_biblioteki as alias
alias.funkcja_z_bilbioteki()
```



Python Wyjątki, klasy, biblioteki
import

from <module_name> import <name(s)>

from <module_name> import *

from <module_name> import <name> as <alt_name>

Python Wyjątki, klasy, biblioteki
Dziedziczenie

```
from math import pi
from math import pi as PI
import math as mh

from math import *  # Niezalecane - może nadpisać istniejące zmienne / funkcje itd.
```

```
dir()
```
zwraca listę nazw dostępnych w bieżącej przestrzeni nazw

```
# Włącz reload on change
%load_ext autoreload
%autoreload 2
```
automatyczne przeładowywanie modułów

Zadanie 8.9 (instrukcja)

1. Stwórz plik .py o nazwie analiza_danych.py
2. W pliku zdefiniuj dwie funkcje: oblicz_srednia i oblicz_mediane.
3. Stwórz nowy plik Jupyter o nazwie test_analizy.py
4. Zaimportuj funkcje ze swojego modułu test_analizy.py
5. W pliku utwórz listę liczb i użyj funkcji z modułu analiza_danych do obliczenia średniej arytmetycznej i mediany.

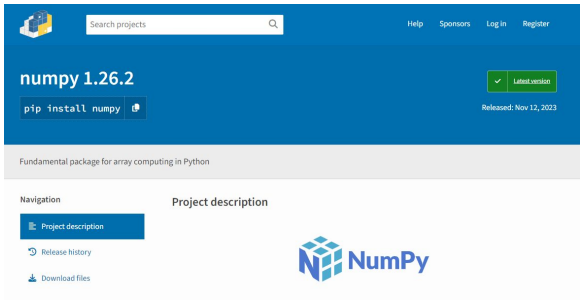
Python Wyjątki, klasy, biblioteki

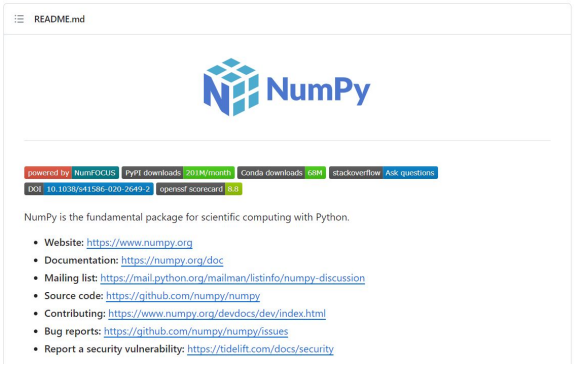
Importowanie modułów - podsumowanie

infoShareAcademy.com

Python Wyjątki, klasy, biblioteki

Biblioteki w Pythonie - dokumentacja



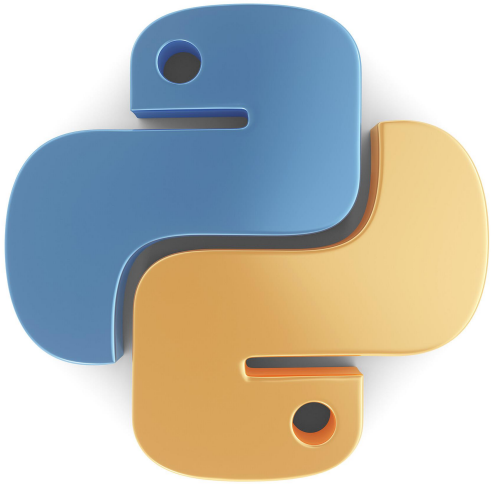


Zadanie 8.10 (instrukcja)

1. Stwórz prosty system bankowy, gdzie klasa KontoBankowe będzie reprezentować konto użytkownika, umożliwiając operacje takie jak wpłata, wypłata i sprawdzenie salda. Ponadto, będzie obsługiwać niestandardowe wyjątki związane z próbą wypłaty większej kwoty niż dostępne saldo na koncie. Po stworzeniu klasy, w notatniku importuj klasę i przetestuj różne scenariusze.

Python Wyjątki, klasy, biblioteki

Podsumowanie



infoShareAcademy.com