

# PyStart #36 Wchodzimy w obiektowość

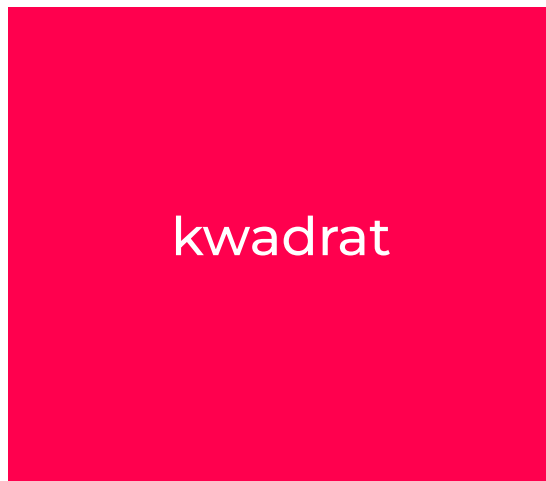
## Pojęcie klasy i obiektu



Co to jest? Co możesz o nim powiedzieć?

# PyStart #54 Wchodzimy w obiektowość

## Pojęcie klasy i obiektu



Co to jest? Co możesz o nim powiedzieć?

# PyStart #54 Wchodzimy w obiektowość

## Pojęcie klasy i obiektu



różowy

długość boku 10

wyśrodkowany

kwadrat

Co to jest? Co możesz o nim powiedzieć?

# PyStart #54 Wchodzimy w obiektowość

## Pojęcie klasy i obiektu



różowy

długość boku 10

wyśrodkowany

kwadrat

mogę obliczyć jego pole

mogę obliczyć jego obwód

mogę go przesunąć / obrócić

Co to jest? Co możesz o nim powiedzieć?

# PyStart #54 Wchodzimy w obiektowość

## Pojęcie klasy i obiektu

### Właściwości / properties

różowy

długość boku 10

wyśrodkowany

kwadrat

### Metody

mogę obliczyć jego pole

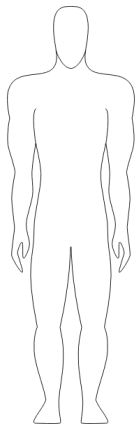
mogę obliczyć jego obwód

mogę go przesunąć / obrócić

Co to jest? Co możesz o nim powiedzieć?

# PyStart #54 Wchodzimy w obiektowość

## Pojęcie klasy i obiektu



klasa



Janek

obiekt

# PyStart #54 Wchodzimy w obiektowość

## Przykładowa klasa



```
1 class Student:
2     def __init__(self, first_name: str, last_name: str, semester: int = 1):
3         self.first_name = first_name
4         self.last_name = last_name
5         self.semester = semester
6
7     def promote(self):
8         self.semester += 1
9
```

# PyStart #54 Wchodzimy w obiektowość

## Przykładowa klasa



```
1 class Student:
2     def __init__(self, first_name: str, last_name: str, semester: int = 1):
3         self.first_name = first_name
4         self.last_name = last_name
5         self.semester = semester
6
7     def promote(self):
8         self.semester += 1
9
```



# PyStart #54 Wchodzimy w obiektowość

## Przykładowa klasa



```
1 class Student:
2     def __init__(self, first_name: str, last_name: str, semester: int = 1):
3         self.first_name = first_name
4         self.last_name = last_name
5         self.semester = semester
6
7     def promote(self):
8         self.semester += 1
9
```

# PyStart #54 Wchodzimy w obiektowość

## Przykładowa klasa



```
1 class Student:
2     def __init__(self, first_name: str, last_name: str, semester: int = 1):
3         self.first_name = first_name
4         self.last_name = last_name
5         self.semester = semester
6
7     def promote(self):
8         self.semester += 1
9
```

# PyStart #54 Wchodzimy w obiektowość

## Przykładowa klasa



```
1 class Student:
2     def __init__(self, first_name: str, last_name: str, semester: int = 1):
3         self.first_name = first_name
4         self.last_name = last_name
5         self.semester = semester
6
7     def promote(self):
8         self.semester += 1
9
```

# PyStart #54 Wchodzimy w obiektowość

## Przykładowa klasa

Podsumowując:

1. Nazwy klas z wielkiej litery
2. `__init__` to metoda specjalna, która jest wywoływana przy tworzeniu obiektu danej klasy oraz służy do przekazywania zmiennych inicjalizujących. **Dwa podkreślenia!**
3. `self` pozwala na ustawienie wartości oraz odebranie wartości konkretnego obiektu.



# PyStart #54 Wchodzimy w obiektowość

## Jak stworzyć obiekt?



```
1 class Student:
2     def __init__(self, first_name: str, last_name: str, semester: int = 1):
3         self.first_name = first_name
4         self.last_name = last_name
5         self.semester = semester
6
7
8
9
10
11     george = Student('George', 'Jetson', 1)
12     print(george)
13     print(george.first_name, george.last_name)
14     george.promote()
15
```

# PyStart #54 Wchodzimy w obiektowość

## Zadania dla nabrania wprawy

1. Zaimplementuj klasę **Circle**, która w metodzie **init** powinna odebrać promień koła. Klasa ta powinna posiadać dwie metody liczące pole, a także obwód koła. **Pamiętaj o testach.**
2. Wykorzystaj utworzoną w poprzednim zadaniu klasę. Zapytaj użytkownika o promień koła i wyświetl jego pole i obwód.
3. Przygotuj klasę **Car**, która powinna przechowywać nazwę samochodu oraz jego cenę i maksymalną prędkość. Zapytaj użytkownika o 5 samochodów, a następnie wyświetl je na ekranie.



# PyStart #55 Łączenie ze sobą obiektów

## Dlaczego warto?

- dzięki klasom aplikacja jest podzielona logicznie
- każda jej część odpowiada za wyspecjalizowaną czynność
- nie musimy martwić się o powtarzające się nazwy funkcji
- w ten sposób lepiej odwzorowujemy rzeczywistość




# PyStart #55 Łączenie ze sobą obiektów

## Jak to działa?




```
6
7 class Library:
8     def __init__(self):
9         self.books = []
10
```



```
1 class Author:
2     def __init__(self, first_name, last_name):
3         self.first_name = first_name
4         self.last_name = last_name
```

```
12 class Book:
13     def __init__(self, title, author):
14         self.title = title
15         self.author = author
16
```





**PyStart #55 Łączenie ze sobą obiektów**

**Słowo na dziś**



# Odpowiedzialność

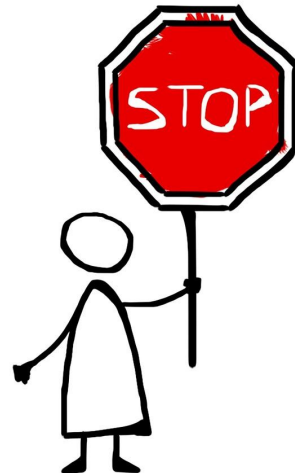


# PyStart #55 Łączenie ze sobą obiektów

## Zadania dla nabrania wprawy

Przygotuj klasy, które odwzorują takie dane, a następnie utwórz obiekty według tej struktury.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <ksiazka>
3    <tytul>Przykładowy tytuł</tytul>
4    <gatunek>Kryminał</gatunek>
5    <autorzy>
6      <autor>
7        <imie>Bonifacy</imie>
8        <nazwisko>Smith</nazwisko>
9        <data_urodzenia>10-10-1910</data_urodzenia>
10     </autor>
11     <autor>
12       <imie>John</imie>
13       <nazwisko>Smith</nazwisko>
14       <data_urodzenia>15-05-1905</data_urodzenia>
15     </autor>
16   </autorzy>
17   <opis>Opis książki</opis>
18   <streszczenie>Krótkie streszczenie</streszczenie>
19   <ocena>5.0</ocena>
20 </ksiazka>
```



# PyStart #55 Łączenie ze sobą obiektów

## Zadania dla nabrania wprawy

Zosia chodziła po łące i zbierała jabłka.

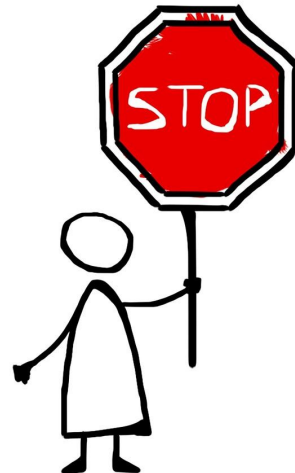
**Każdy owoc ma:**

**kolor** np. czerwony, zielony, żółty,

**smak** np. słodki, kwaśny

**rodzaj** np. dojrzały, niedojrzały.

Przygotuj obiekt Koszyk, do którego można będzie dodawać owoce oraz generować raport według danej cechy. np. ilość owoców według kolorów. Idealnie gdyby raport również był obiektem :)



# PyStart #55 Łączenie ze sobą obiektów

## Zadania dla nabrania wprawy

Przygotuj program, który będzie przechowywał wszystkie Twoje spotkania.  
Informacje o spotkaniach przechowuj w pliku meetings.json.

### Wymagania:

Każde ze spotkań trwa zawsze godzinę.

Spotkania mogą mieć różne tytuły.

Na dany dzień, na daną godzinę może być zapisane tylko jedno spotkanie.

Kalendarz posiada metody umożliwiające:

- wyświetlanie wszystkich spotkań
- sprawdzenie czy dany termin(data i godzina) jest wolna



# PyStart #56 Inne metody specjalne

## Kilka informacji

- Dlaczego metoda specjalna?
- Pamiętaj o `__` `__` :-)
- Czy jest więcej metod niż `__init__` ?



# PyStart #56 Inne metody specjalne

## Metody specjalne

`__init__` - utworzenie obiektu

`__str__` - reprezentacja tekstowa

`__add__` - dodanie dwóch obiektów (tak jak list)

`__eq__` - porównanie dwóch obiektów `==`

`__ge__` - porównanie obiektów `>=`

`__le__` - porównanie obiektów `<=`



# PyStart #56 Inne metody specjalne

## Metody specjalne `__str__`



```
1 class Person:
2     def __init__(self, first_name: str, last_name: str):
3         self.first_name = first_name
4         self.last_name = last_name
5
6     def __str__(self):
7         return f'{self.first_name} {self.last_name}'
8
9
10 me = Person('Kacper', 'Sieradziński')
11 print(me)
12
```

# PyStart #56 Inne metody specjalne

## Metody specjalne `__add__`



```
1 class Box:
2     def __init__(self, capacity: int):
3         self.capacity = capacity
4
5     def __add__(self, other):
6         total = self.capacity + other.capacity
7         return Box(total)
```

```
10 box1 = Box(10)
11 box2 = Box(20)
12 box3 = box1 + box2
13 print(box3.capacity)
```



# PyStart #56 Inne metody specjalne

## Metody specjalne `__eq__`



```
1 class Box:
2     def __init__(self, capacity: int):
3         self.capacity = capacity
4
5     def __eq__(self, other):
6         return self.capacity == other.capacity
7
8
9 box1 = Box(10)
10 box2 = Box(10)
11 print(box1 == box2)
12
```

# PyStart #56 Inne metody specjalne

## Metody specjalne

`__lt__` - mniejsze niż

`__gt__` - większe niż

`__ne__` - nierówne



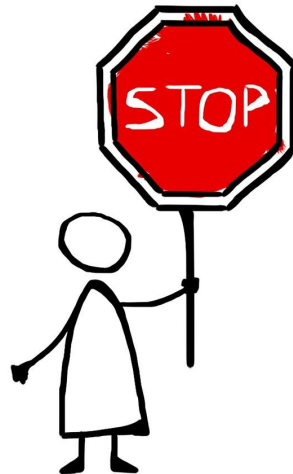
# PyStart #56 Inne metody specjalne

## Zadania dla nabrania wprawy

Dany jest obiekt:

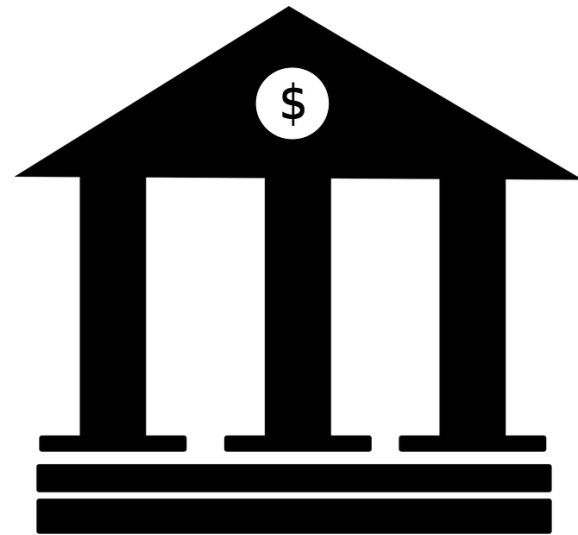
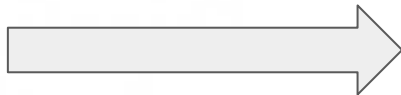
```
1 class LengthUnit:
2     def __init__(self, value: int):
3         self.value = value
4
5     def to_centimeter(self):
6         return self.value / 10
7
8     def to_meter(self):
9         return self.value / 10 / 100
10
```

Przygotuj metody specjalne umożliwiające dodawanie obiektów do siebie, odejmowanie oraz porównywanie ich ze sobą.



# PyStart #57 Enkapsulacja

## O co chodzi?



# PyStart #57 Enkapsulacja

## Czego używamy zamiast?



# PyStart #57 Enkapsulacja

## O co chodzi?




**Enkapsulacja** - ukrywanie pól danej klasy dla innych klas. W ten sposób chronimy siebie i innych programistów przed **nieprzewidzianym przez nas** modyfikowaniem właściwości lub wywoływaniem metod.

# PyStart #57 Enkapsulacja

## Enkapsulacja w Pythonie


```
1 class Bank:
2     def __init__(self):
3         self._money = 1000 * 1000
4
5     def withdraw(self, amount: int):
6         if amount > self._money:
7             return 0
8
9         self._money -= amount
10
11        return amount
12
13    def deposit(self, amount: int):
14        self._money += amount
```

- 
- Symbol podkreślenia przed nazwą zmiennej lub metody.
  - Pycharm nie podpowiada takich zmiennych.
  - Nie chodzi tu o bezpieczeństwo, ale o wygodę

# PyStart #57 Enkapsulacja

## Enkapsulacja w Pythonie

```
1 class Bank:
2     def __init__(self):
3         self._money = 1000 * 1000
4
5     def _convert_to_pln(self, amount):
6         ...
7
```

- 
- Metody także mogą być “prywatne”
  - Będzie można wywoływać je wewnątrz klasy poprzez `self._convert_to_pln`, a z “zewnątrz” już nie.



# PyStart #57 Enkapsulacja

## Enkapsulacja w Pythonie



A co jeśli spróbujemy się do tego odwołać mimo tego?

# PyStart #57 Enkapsulacja

## Enkapsulacja w Pythonie



A co jeśli spróbujemy się do tego odwołać mimo tego?

**Zadziała zwyczajnie, ale sprawdźmy to.**

# PyStart #57 Enkapsulacja

## Zadania dla nabrania wprawy

Przygotuj program, który będzie przechowywał listę zakupów. Każdy wpis jest osobnym obiektem klasy `ListItem`. Jeśli dany produkt znajduje się już na liście nie powinien być dodany drugi raz, zamiast tego powinna być zwiększana jego ilość. Produkty do zakupienia przechowuj w zmiennej prywatnej.

Obiekt klasy `List` musi posiadać następujące metody:

`addItem(product: Product, quantity: float)`

`removeItem(product: Product, quantity: float)`

`listItems(): list`

`calculateTotalCost()`

Każdy obiekt klasy `Product` posiada cenę oraz nazwę



# PyStart #58 Dziedziczenie

O co chodzi?

## Pojazdy

Samochody

Spalinowe



Elektryczne



Motocykle



Łodzie



# PyStart #58 Dziedziczenie

## Jak to wygląda w kodzie?

```
1 class Vehicle:
2     pass
3
4
5 class BaseCar(Vehicle):
6     pass
7
8
9 class Car(BaseCar):
10    pass
11
12
13 class ElectricCar(BaseCar):
14    pass
15
16
```



- W nawiasie przekazujemy po jakiej klasie dziedziczymy
- Dziedziczenie przebiega “kaskadowo”, dziedziczone są właściwości i metody

# PyStart #58 Dziedziczenie

## Przeciążanie metod

```
1 class Parent:
2     def get_type(self):
3         return 'parent'
4
5
6 class Child(Parent):
7     def get_type(self):
8         return 'child'
9
10
11 sample = Child()
12 print(sample.get_type())
13
```

→ Przeciążanie metod polega na ich “nadpisywaniu”, tzn. w klasie potomnej posiadamy taką samą metodę jak w “rodzicu”

```
(venv) D:\Trainings\Pystart\week_7>python inheritance.py
child
```

# PyStart #58 Dziedziczenie

## Przeciążanie metod



A co jeśli potrzebowalbyś odebrać wartość z metody “rodzica” ?

**Super!**

# PyStart #58 Dziedziczenie

## Super()



```
1 class Product:
2     def __init__(self, price):
3         self.price = price
4
5     def get_price(self):
6         return self.price
7
8
9 class DiscountedProduct(Product):
10    def get_price(self):
11        price = super().get_price()
12        return price - 0.1 * price
13
14
15 product = DiscountedProduct(100)
16 print(product.get_price())
```

→ **super()** wskazuje na klasę z której dziedziczymy, którą przeciążamy.



# PyStart #58 Dziedziczenie

## Kiedy super, a kiedy nie super ?



```
1 class Parent:
2     def parent_method(self):
3         return 'parent_method'
4
5     def common_method(self):
6         return 'common method'
7
8
9 class Child(Parent):
10     def child_method(self):
11         return self.parent_method()
12
13     def common_method(self):
14         return super().common_method()
15
```

- Jeśli ta sama metoda (**common\_method**) jest dostępna w obu klasach (**Parent i Child**), to potrzebujemy `super()`
- Jeśli metody mają różne nazwy to w `Child` można wywoływać metody z `self`.  
**self.parent\_method()**

# PyStart #58 Dziedziczenie

## Jak zachowa się `__init__` ?



```
1  class Parent:
2      def __init__(self):
3          print('Rodzic!')
4
5
6  class Child(Parent):
7      pass
8
9
10 sample = Child()
11
```


- Init zachowuje się tak samo jak każda inna metoda.
- Init zadziałał, komunikat się wyświetla.

```
(venv) D:\Trainings\Pystart\week_7>python inheritance.py
Rodzic!
```

# PyStart #58 Dziedziczenie

## Przeciążanie inita

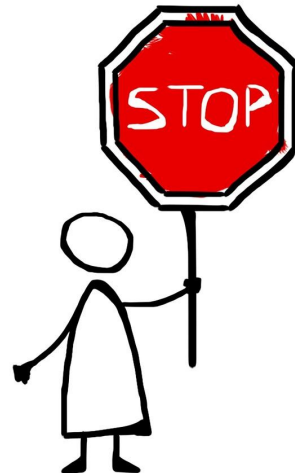
```
1 class Person:
2     def __init__(self, first_name, last_name):
3         self.details = f'{first_name} {last_name}'
4
5
6 class Student(Person):
7     def __init__(self, first_name, last_name, semester):
8         super().__init__(first_name, last_name)
9         self.semester = semester
10
11
12 jan = Student('Jan', 'Kowalski', 2)
13 print(jan.details)
14 print(jan.semester)
15
```

- 
- Przeciążanie inita jest możliwe tak samo jak każdej innej metody.
  - Koniecznie należy wówczas wywołać init klasy po której dziedziczymy.

# PyStart #58 Dziedziczenie

## Zadania dla nabrania wprawy

1. Przygotuj klasę **BankAccount**, która będzie pozwalała wpłacać(**deposit**) i wypłacać(**withdraw**) środki. Utwórz klasę konta oszczędnościowego **SavingsAccount**, która będzie dziedziczyła po **BankAccount** i umożliwi zwiększenie stanu konta o procent odsetek.
2. Przygotuj klasę **Employee**, która w inicie będzie odbierała imię, nazwisko oraz stawkę godzinową. Przygotuj klasę **Manager**, która będzie dziedziczyła po klasie **Employee**, którego każda godzina pracy będzie liczona podwójnie, a dodatkowo będzie możliwość określenia premii managera(**add\_bonus**(amount: int)).



# PyStart #59 Wyjątki, obsługa błędów

Co może pójść nie tak?



```
1
2     number1 = int(input('Number 1: '))
3     number2 = int(input('Number 2: '))
4
5     result = number2 / number1
6     print(result)
```

**2, 3:** Input może zwrócić coś czego int nie będzie mógł zrobić liczby.

**5:** Nie można dzielić przez zero

# PyStart #59 Wyjątki, obsługa błędów

Co może pójść nie tak?



```
1
2     number1 = int(input('Number 1: '))
3     number2 = int(input('Number 2: '))
4
5     result = number2 / number1
6     print(result)
```

**2, 3:** Input może zwrócić coś czego int nie będzie mógł zrobić liczby.

**ValueError**

**5:** Nie można dzielić przez zero

**ZeroDivisionError**

# PyStart #59 Wyjątki, obsługa błędów

## Jak to zapisać bezpieczniej?



```
1 try:
2     number1 = int(input('Number 1: '))
3     number2 = int(input('Number 2: '))
4     result = number2 / number1
5     print(result)
6 except ValueError:
7     print('Podano nieprawidłową wartość')
8 except ZeroDivisionError:
9     print('Nie dzielimy przez zero')
```

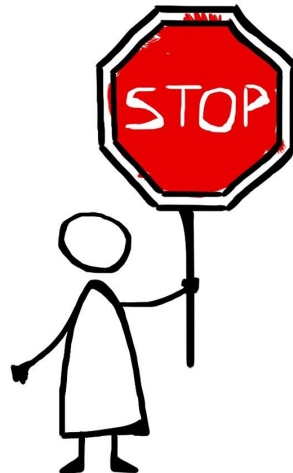
- Try... except
- Ile razy ten except?
- można też  
`except (ValueError, ZeroDivisionError)`
- finally

# PyStart #59 Wyjątki, obsługa błędów

## Zadania dla nabrania wprawy

Przygotuj listę która zawiera 10 elementów. Zapytaj użytkownika o to która z wartości powinna zostać zwrócona. Jeśli użytkownik odwoła się do wartości która nie istnieje (**IndexError**), to wyświetl komunikat, że takiej wartości nie ma.

Przygotuj słownik zawierający 3 wartości. Sprawdź co się stanie jeśli spróbujesz odwołać się od klucza, który nie istnieje w tym słowniku. Spróbuj obsłużyć taką sytuację łapiąc taki wyjątek.





# PyStart #59 Wyjątki, obsługa błędów

## Jak wysłać wyjątek?



Jak wyrzucić wyjątek w przypadku gdy podana liczba nie jest podzielna przez 5?

```
1 while True:
2     value = int(input('Podaj liczbę: '))
3     if not value % 5 == 0:
4         raise Exception('Liczba nie jest podzielna przez 5.')
5
6     print(value)
```

Traceback (most recent call last):

File "exceptions-show.py", line 4, in <module>

raise Exception('Liczba nie jest podzielna przez 5.')

Exception: Liczba nie jest podzielna przez 5.

# PyStart #59 Wyjątki, obsługa błędów

## Ciekawi?



Gdzie należy dodać try except?

**To zależy.**

# PyStart #59 Wyjątki, obsługa błędów

## Jak wysłać wyjątek?

Jak wyrzucić wyjątek w przypadku gdy podana liczba nie jest podzielna przez 5?

```
1 while True:
2     try:
3         value = int(input('Podaj liczbę: '))
4         if not value % 5 == 0:
5             raise Exception('Liczba nie jest podzielna przez 5.')
6
7         print(value)
8     except Exception as e:
9         print(e)
```

#1

# PyStart #59 Wyjątki, obsługa błędów

## Jak wysłać wyjątek?

Jak wyrzucić wyjątek w przypadku gdy podana liczba nie jest podzielna przez 5?

```
1 while True:
2     try:
3         value = int(input('Podaj liczbę: '))
4         if not value % 5 == 0:
5             raise Exception('Liczba nie jest podzielna przez 5.')
6
7         print(value)
8     except Exception as e:
9         print(e)
```

#1

#2

```
1 try:
2     while True:
3         value = int(input('Podaj liczbę: '))
4         if not value % 5 == 0:
5             raise Exception('Liczba nie jest podzielna przez 5.')
6
7         print(value)
8     except Exception as e:
9         print(e)
10
```

# PyStart #59 Wyjątki, obsługa błędów

## Zadania dla nabrania wprawy

Zapytaj użytkownika o dowolny tekst, który użytkownik chciałby odwrócić  
(Wyświetlić od końca do początku).

Jeśli użytkownik poda pusty string powinien zostać zwrócony wyjątek.

Pytaj użytkownika o nazwy 10 owoców, jeżeli dany owoc znajduje się już na liście to powinien zostać wyrzucony wyjątek i program powinien przestać pytać.



# PyStart #59 Wyjątki, obsługa błędów

## Ciekawi?



```
1 class ValueTooSmall(Exception):  
2     pass  
3  
4 class ValueTooBig(Exception):  
5     pass  
6
```

```
8 try:  
9     value = int(input('Podaj liczbę: '))  
10    if value < 10:  
11        raise ValueTooSmall()  
12    if value > 20:  
13        raise ValueTooBig()  
14  
15    print(value)  
16 except (ValueTooSmall, ValueTooBig) as error:  
17    print(error)
```

# PyStart #59 Wyjątki, obsługa błędów

## Podsumowując

- Wyjątki służą do obsługi sytuacji gdy wystąpi błąd.
- Różne typy błędów realizujemy za pomocą dziedziczenia.
- W Pythonie jest wiele “wbudowanych” typów wyjątków, warto zwrócić uwagę na komunikaty.



# PyStart #59 Wyjątki, obsługa błędów

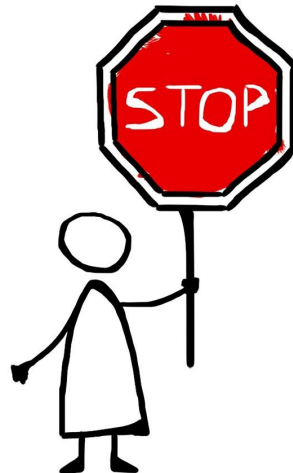
## Zadania dla nabrania wprawy

Na podstawie API <https://restcountries.com> Przygotuj aplikację, która zapyta użytkownika o nazwę Państwa, a następnie wyświetli jego stolicę.

Pamiętaj by obsłużyć sytuację gdy:

- użytkownik wprowadzi nazwę Państwa, którego nie ma w bazie.
- skrypt nie będzie miał dostępu do internetu

Warto zastanowić się również nad tym co powinno się stać gdy użytkownik zada pytanie o to samo Państwo dwa razy. **Czy powinniśmy pytać o nie drugi raz API? Pomyśl o uzasadnieniu swojej decyzji.**





# DZIEWIĄTA PRACA DOMOWA

## UWAGA! UWAGA!



- Stwórz klasę `BaseEmployee` umożliwiającą przechowywanie imienia, nazwiska oraz daty zatrudnienia. Jeśli użytkownik poda datę późniejszą niż data dzisiejsza lub wskazującą na czas pracy dłuższy niż 50 lat to wyrzucić wyjątek `InvalidDateOfEmployment`. Dodaj metody specjalne umożliwiające sortowanie względem długości zatrudnienia.
- Utwórz w klasie `BaseEmployee` metodę `get_employment_time`. Metoda ta powinna każdorazowo zwracać informację ile dni pracuje dany pracownik(od początku).
- Utwórz klasę `Employee` która będzie dziedzyczyła po klasie `BaseEmployee`, klasa ta powinna posiadać stawkę godzinową, wymiar etatu (np. pełen etat to 160 godzin), oraz wartość premii pracownika. Przygotuj metodę wyliczającą wartość jego wynagrodzenia.
- Napisz klasę obsługującą plik json wg. wzoru.  
Powinna ładować zawartość pliku json do zmiennej, a jeśli plik nie istnieje zmienna powinna zawierać pustą listę. Dodatkowo powinna mieć możliwość dodawania nowego obiektu do pliku

```
[  
  { "first_name": "James", "last_name": "Callaghan" },  
  { "first_name": "Margaret", "last_name": "Thatcher" },  
  { "first_name": "James", "last_name": "Callaghan" },  
  { "first_name": "Harold", "last_name": "Wilson" },  
]
```