Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny

Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia

Rok akademicki 2024/2025

Wykład nr 6 (13.05.2025)

dr inż. Jarosław Forenc

Plan wykładu nr 6

- Programowanie obiektowe
  - przykład (obwód RLC), liczby zespolone
  - dziedziczenie
  - dziedziczenie wielokrotne
  - prawa dostępu
- Biblioteka standardowa Pythona
  - wbudowane funkcje
  - wbudowane stałe
  - wbudowane typy
  - wbudowane wyjątki
  - moduły (string, datetime, zoneinfo, calendar)

Python - przykład (klasa SerialRLC)

```
import math

class SerialRLC:
    def __init__(self, R, L, C):
        self.R = R  # Rezystancja (Ohm)
        self.L = L  # Indukcyjność (Henr)
        self.C = C  # Pojemność (Farad)

    def __str__(self):
        return f"Obwód RLC: R = {self.R} Ω, L = {self.L}
                 H, C = {self.C} F"

    def resonant_frequency(self):
        # Częstotliwość rezonansowa (Hz)
        if self.L > 0 and self.C > 0:
            return 1 / (2 * math.pi * math.sqrt(self.L
                   * self.C))
        else:
            return None  # Dla nieprawidłowych danych
```

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

Python - przykład (klasa SerialRLC)

```
    def impedance(self, f):
        # Impedancja zespolona przy częstotliwości f (Hz)
        omega = 2 * math.pi * f
        XL = omega * self.L
        XC = 1 / (omega * self.C)
        Z = complex(self.R, XL - XC)
        return abs(Z)

    def phase_angle(self, f):
        # kąt (w stopniach) przy częstotliwości f (Hz)
        omega = 2 * math.pi * f
        XL = omega * self.L
        XC = 1 / (omega * self.C)
        angle_rad = math.atan2(XL - XC, self.R)
        return math.degrees(angle_rad)
```

$$|Z| = \sqrt{R^2 + \left(\omega L - \frac{1}{\omega C}\right)^2}$$

$$\phi = \arctan\left(\frac{\omega L - \frac{1}{\omega C}}{R}\right)$$

Python - przykład (klasa SerialRLC)

```
obwod = SerialRLC(R=100, L=0.5, C=1e-6)
print(obwod)

fr = obwod.resonant_frequency()
print(f"Częstotliwość rezonansowa: {fr:.2f} Hz")

f_test = 1000  # 1 kHz
Z = obwod.impedance(f_test)
angle = obwod.phase_angle(f_test)
print(f"Impedancja przy {f_test} Hz: {Z:.2f} Ω")
print(f"Kąt fazowy przy {f_test} Hz: {angle:.2f}°")
```

```
Obwód RLC: R = 100 Ω, L = 0.5 H, C = 1e-06 F
Częstotliwość rezonansowa: 225.08 Hz
Impedancja przy 1000 Hz: 2984.11 Ω
Kąt fazowy przy 1000 Hz: 88.08°
```

Python - liczby zespolone

- do tworzenia liczb zespolonych stosujemy literę „j” lub funkcję complex()

```
z1 = 2 + 5j
z2 = complex(-3,2.5)

print("z1 =",z1)
print("z2 =",z2)
```

```
z1 = (2+5j)
z2 = (-3+2.5j)
```

- część rzeczywista (real) i część urojona (imag) liczby zespolonej:

```
z1 = 2 + 5j

print("Re{z1} =",z1.real)
print("Im{z1} =",z1.imag)
```

```
Re{z1} = 2.0
Im{z1} = 5.0
```

Python - liczby zespolone

- moduł i kąt liczby zespolonej:

```
import cmath

z1 = 2 + 5j

modul = abs(z1)
kat_rad = cmath.phase(z1)
kat_deg = kat_rad * 180 / cmath.pi

print(f"Moduł:              {modul:.4f}")
print(f"Kąt (w radianach): {kat_rad:.2f}")
print(f"Kąt (w stopniach): {kat_deg:.2f}")
```

```
Moduł:              5.3852
Kąt (w radianach): 1.19
Kąt (w stopniach): 68.20
```

Python - liczby zespolone

- operacje arytmetyczne:

```
z1 = 2 + 4j
z2 = 6 - 8j
print("z1 =",z1)
print("z2 =",z2)

z3 = z1 + z2
print("z3 = z1 + z2 =",z3)

z3 = z1 - z2
print("z3 = z1 - z2 =",z3)

z3 = z1 * z2
print("z3 = z1 * z2 =",z3)

z3 = z1 / z2
print("z3 = z1 / z2 =",z3)
```

```
z1 = (2+4j)
z2 = (6-8j)
z3 = z1 + z2 = (8-4j)
z3 = z1 - z2 = (-4+12j)
z3 = z1 * z2 = (44+8j)
z3 = z1 / z2 = (-0.2+0.4j)
```

Python - moduł cmath

- Stałe i funkcje matematyczne dostępne w module o nazwie cmath
- Aby korzystać z tych funkcji i stałych, należy najpierw zaimportować ten moduł

```
import cmath
```

- Stałe matematyczne:
  - cmath.pi - stała pi (π)
  - cmath.tau - wartość $2 \cdot \pi$
  - cmath.e - liczba Eulera, stała e (podstawa logarytmu naturalnego)
  - cmath.inf - nieskończoność
  - cmath.nan - NaN (Not a Number) - reprezentuje wartość nieokreśloną lub niemożliwą do reprezentacji numerycznej

Python - moduł cmath

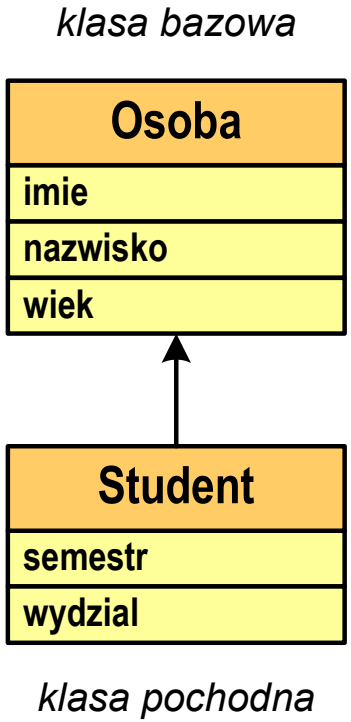
- Funkcje matematyczne:
  - cmat.sqrt(x) - pierwiastek kwadratowy liczby zespolonej (także z liczb ujemnych)
  - cmath.exp(x) - wartość funkcji wykładniczej e^x
  - cmath.log(x) - logarytm naturalny ln(x)
  - cmath.log10(x) - logarytm dziesiętny
  - cmath.sin(x) - sinus liczby zespolonej
  - cmath.cos(x) - cosinus liczby zespolonej
  - cmath.tan(x) - tangens liczby zespolonej
  - cmath.asin(x) - arcus sinus (odwrotność sinusa)
  - cmath.acos(x) - arcus cosinus
  - cmath.atan(x) - arcus tangens

Python - moduł cmath

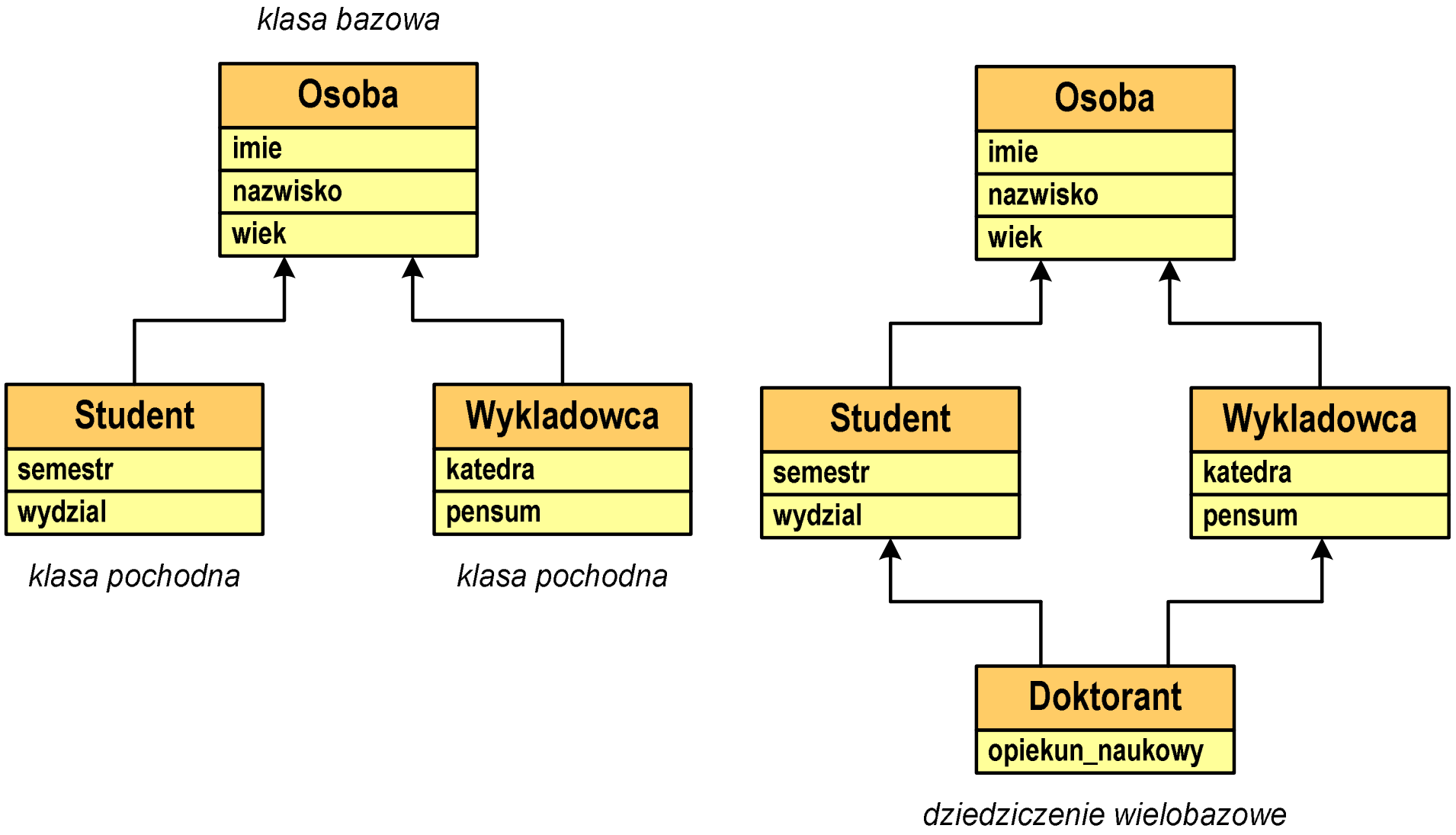
- Funkcje matematyczne:
  - cmath.phase(x) - kąt (faza) liczby zespolonej w radianach
  - cmath.polar(x) - zwraca moduł i fazę jako krotkę: (r, phi)
  - cmath.rect(r, phi) - tworzy liczbę zespoloną z modułu r i kąta phi (czyli postać trygonometryczna)
  - cmath.isclose(a, b) - porównuje dwie liczby zespolone z tolerancją
  - cmath.isnan(x) - sprawdza, czy liczba zespolona zawiera NaN
  - cmath.isinf(x) - sprawdza, czy liczba zespolona zawiera nieskończoność

Python - prog. obiektowe (dziedziczenie)

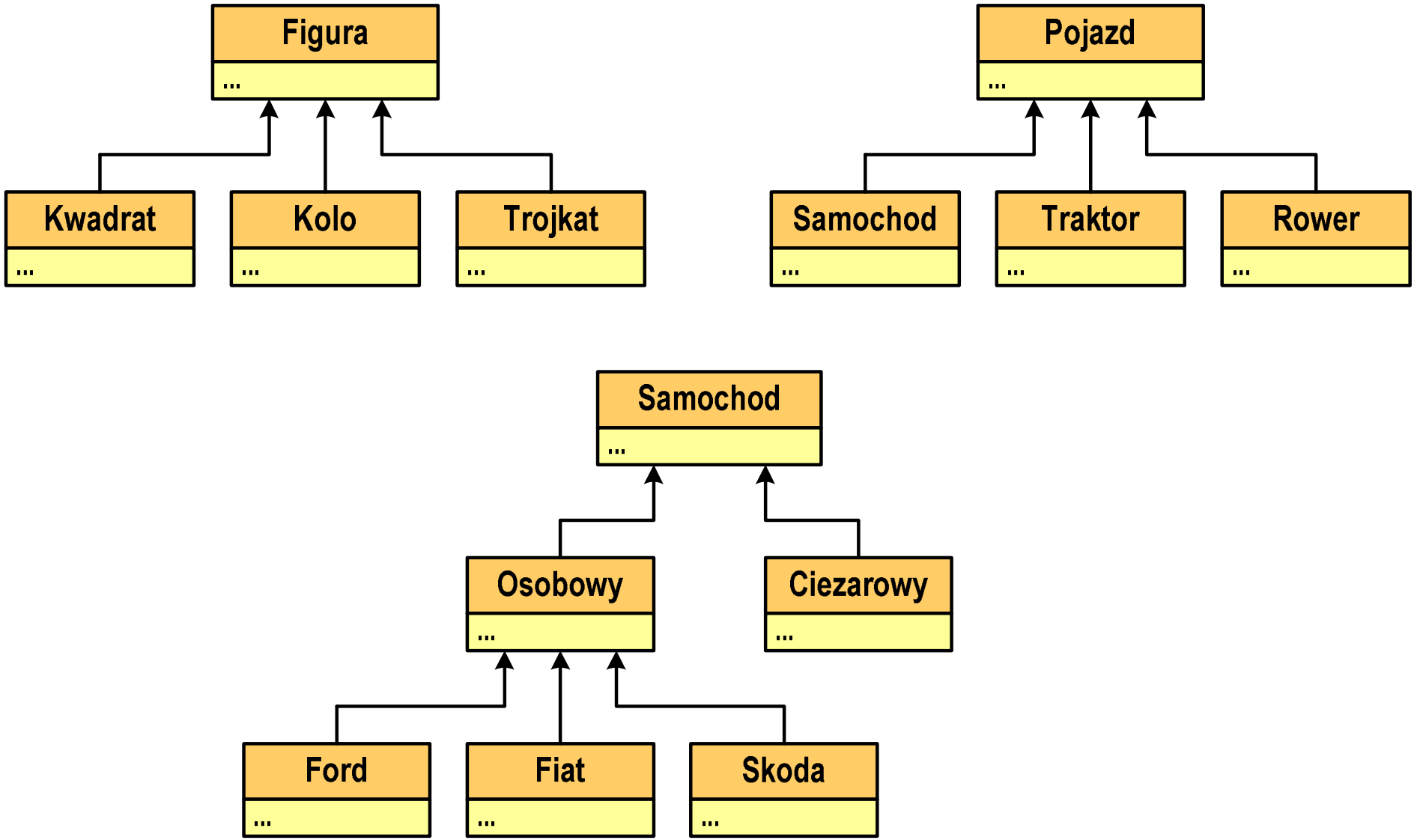
- dziedziczenie to technika, która pozwala na tworzenie nowych klas na podstawie klas już istniejących
- klasa, z której dziedziczą inne klasy, nazywana jest klasą bazową (podstawową, nadrzędną, superklasą)
- nowa klasa nazywa się klasą pochodną (potomną, subklasą)
- nowa klasa automatycznie dziedziczy atrybuty i metody klasy bazowej
- w klasie pochodnej można definiować również nowe atrybuty i metody



Python - prog. obiektowe (dziedziczenie)



Python - prog. obiektowe (dziedziczenie)



Python - prog. obiektowe (dziedziczenie)

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

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

class Student(Osoba):
    def __init__(self, imie, nazwisko, semestr, wydział):
        super().__init__(imie, nazwisko)
        self.semestr = semestr
        self.wydział = wydział

    def __str__(self):
        return (f"Student: {self.imie} {self.nazwisko}, "
                f"semestr: {self.semestr}, wydział: "
                f"{self.wydział}")
```

Python - prog. obiektowe (dziedziczenie)

- klasa bazowa musi znajdować się w tym samym pliku, w którym tworzymy klasę pochodną
- nazwa klasy bazowej musi być umieszczona w nagłówku klasy pochodnej

```
class Student(Osoba):
```

- w metodzie __init__() klasy pochodnej wywołujemy metodę __init__() z klasy bazowej

```
super().__init__(imie, nazwisko)
```

- super() jest to wbudowana funkcja, która zwraca obiekt specjalny (zwany delegatem), umożliwiający dostęp do metod i atrybutów klasy bazowej z wnętrza klasy pochodnej
- super() stosowane jest do wywołania konstruktora lub metod klasy bazowej, nazwa funkcji pochodzi od superklasy

Python - prog. obiektowe (dziedziczenie)

- utworzenie obiektów i wywołanie metod

```
osoba = Osoba("Jan", "Kowalski")
student = Student("Anna", "Nowak", 3, "WE")

print(osoba)
print(student)

print(f"Imię: {student.imie}")
print(f"Nazwisko: {student.nazwisko}")
print(f"Semestr: {student.semestr}")
print(f"Wydział: {student.wydział}")
```

```
Jestem: Jan Kowalski
Student: Anna Nowak, semestr: 3, wydział: WE
Imię: Anna
Nazwisko: Nowak
Semestr: 3
Wydział: WE
```

Python - prog. obiektowe (dziedziczenie)

- istnieje możliwość nadpisania dowolnej metody klasy bazowej

```
class Pojazd:
    def __init__(self, marka, model):
        self.marka = marka
        self.model = model

    def opis(self):
        return f"Pojazd marki {self.marka}, model {self.model}"

class Samochod(Pojazd):
    def __init__(self, marka, model, rok):
        super().__init__(marka, model)
        self.rok = rok

    def opis(self):
        podstawowy_opis = super().opis()
        return f"{podstawowy_opis}, rok produkcji: {self.rok}"
```

Python - prog. obiektowe (dziedziczenie)

- metoda z klasy pochodnej przesłania widoczność metody z klasy bazowej

```
pojazd = Pojazd("Toyota", "Corolla")
samochod = Samochod("Ford", "Mustang", 2022)

print(pojazd.opis())
print(samochod.opis())
```

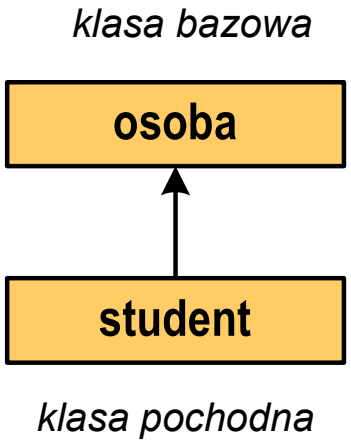
```
Pojazd marki Toyota, model Corolla
Pojazd marki Ford, model Mustang, rok produkcji: 2022
```

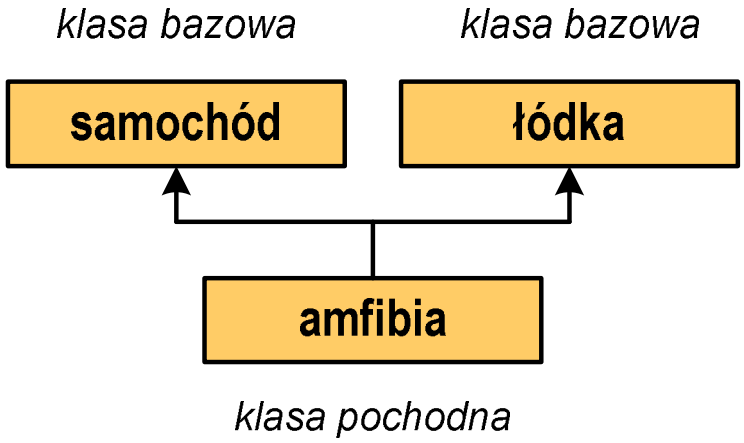
- korzystając z funkcji super() można odwołać się do metody z klasy bazowej

```
def opis(self):
    podstawowy_opis = super().opis()
    return f"{podstawowy_opis}, rok produkcji: {self.rok}"
```

Python - prog. obiektowe (dziedziczenie)

- dziedziczenie jednokrotne (single inheritance) - klasa pochodna dziedziczy po jednej klasie bazowej
- dziedziczenie wielokrotne (multiple inheritance) - klasa pochodna dziedziczy po więcej niż jednej klasie bazowej





Python - prog. obiektowe (dziedziczenie wielokrotne)

```
class A:
    def __init__(self):
        self.cecha_a = "Cecha z klasy A"

    def metoda_a(self):
        return "Metoda z klasy A"

class B:
    def __init__(self):
        self.cecha_b = "Cecha z klasy B"

    def metoda_b(self):
        return "Metoda z klasy B"

class C(A, B):
    def __init__(self):
        A.__init__(self)
        B.__init__(self)
        self.cecha_c = "Cecha z klasy C"

    def metoda_c(self):
        return "Metoda z klasy C"
```

Python - prog. obiektowe (dziedziczenie wielokrotne)

```
obiekt = C()
print(obiekt.cecha_a)
print(obiekt.cecha_b)
print(obiekt.cecha_c)

print(obiekt.metoda_a())
print(obiekt.metoda_b())
print(obiekt.metoda_c())
```

```
Cecha z klasy A
Cecha z klasy B
Cecha z klasy C
Metoda z klasy A
Metoda z klasy B
Metoda z klasy C
```

Python - prog. obiektowe (prawa dostępu)

- w Pythonie nie ma formalnych modyfikatorów dostępu, takich jak np. w języku C++ (public, protected, private)
- zamiast tego stosowane są konwencje nazewnicze i pewne mechanizmy do oznaczania poziomów dostępu do atrybutów i metod w klasach
- domyślnie wszystkie atrybuty i metody są publiczne - oznacza to, że są one dostępne z zewnątrz klasy; nie ma specjalnych prefiksów w nazwach

```
class Klasa:
    def __init__(self):
        self.publiczny_atrybut = "Dostęp publiczny"

    def publiczna_metoda(self):
        return "To jest metoda publiczna"

obiekt = Klasa()
print(obiekt.publiczny_atrybut)  # Dostęp publiczny
print(obiekt.publiczna_metoda())  # To jest metoda publiczna
```

Python - prog. obiektowe (prawa dostępu)

- atrybuty i metody chronione (protected) są oznaczane przez jeden znak podkreślenia _ na początku nazwy
- jest to konwencja wskazująca, że atrybut lub metoda nie powinny być używane na zewnątrz klasy lub jej podklas, choć technicznie jest to możliwe

```
class Klasa:
    def __init__(self):
        self._chroniony_atrybut = "Dostęp chroniony"

    def _chroniona_metoda(self):
        return "To jest metoda chroniona"

obiekt = Klasa()
print(obiekt._chroniony_atrybut)  # Dostęp możliwy, niezalecany
print(obiekt._chroniona_metoda())  # Dostęp możliwy, niezalecany
```

Python - prog. obiektowe (prawa dostępu)

- atrybuty i metody prywatne (private) są oznaczane przez dwa znaki podkreślenia __ na początku nazwy
- Python stosuje name mangling, aby utrudnić dostęp do tych elementów z zewnątrz klasy; Python zmienia nazwę atrybutu, dodając przedrostek _NazwaKlasy

```
class Klasa:
    def __init__(self):
        self.__prywatny_atrybut = "Dostęp prywatny"

    def __prywatna_metoda(self):
        return "To jest metoda prywatna"

obiekt = Klasa()
print(obiekt.__prywatny_atrybut)
print(obiekt.__prywatna_metoda())
```

Python - prog. obiektowe (prawa dostępu)

- atrybuty i metody prywatne (private) są oznaczane przez dwa znaki podkreślenia __ na początku nazwy
- Python stosuje name mangling, aby utrudnić dostęp do tych elementów z zewnątrz klasy; Python zmienia nazwę atrybutu, dodając przedrostek _NazwaKlasy

```
class Klasa:
    def __init__(self):
        self.__prywatny_atrybut = "Dostęp prywatny"

    def __prywatna_metoda(self):

obiekt
print
print
```

```
Traceback (most recent call last):
  File "d:\tempCodeRunnerFile.py", line 9, in <module>
    print(obiekt.__prywatny_atrybut)
          ^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'Klasa' object has no attribute
'__prywatny_atrybut'. Did you mean:
'_Klasa__prywatny_atrybut'?
```

Python - prog. obiektowe (prawa dostępu)

- można uzyskać dostęp do prywatnych atrybutów i metod przez name mangling, ale nie jest to zalecane

```
class Klasa:
    def __init__(self):
        self.__prywatny_atrybut = "Dostęp prywatny"

    def __prywatna_metoda(self):
        return "To jest metoda prywatna"

obiekt = Klasa()

# Name mangling:
print(obiekt._Klasa__prywatny_atrybut)
print(obiekt._Klasa__prywatna_metoda())
```

```
Dostęp prywatny
To jest metoda prywatna
```

Python - biblioteka standardowa

- zbiór modułów i pakietów, które są dystrybuowane wraz z główną instalacją języka Python
- opis: https://docs.python.org/3/library/index.html (EN)
- biblioteka jest bardzo obszerna, zawiera wiele funkcji i modułów, które są przetestowane i wydajne
- biblioteka zawiera moduły:
  - wbudowane, napisane w języku C - dostęp do funkcjonalności systemowych, np. operacje na plikach
  - napisane w Pythonie - standardowe rozwiązania wielu problemów występujących podczas codziennego programowania
- w przypadku systemu Windows instalator Pythona zawiera całą bibliotekę standardową oraz wiele dodatkowych komponentów
- w systemach Linux/Unix, Python jest dostarczany jako zbiór pakietów, więc konieczne może być doinstalowanie niektórych komponentów

Python - biblioteka standardowa

- w skład biblioteki standardowej wchodzą:
  - wbudowane funkcje, wbudowane stałe
  - wbudowane typy, wbudowane wyjątki
  - moduły
- przykładowe przeznaczenie modułów:
  - usługi przetwarzania tekstu i danych binarnych, typy danych
  - moduły numeryczne i matematyczne
  - dostęp do plików i katalogów, kompresja i archiwizacja danych
  - usługi kryptograficzne, ogólne usługi systemu operacyjnego
  - sieci i komunikacja międzyprocesowa, obsługa danych internetowych
  - protokoły internetowe i wsparcie, usługi multimedialne
  - internacjonalizacja, narzędzia developerskie, importowanie modułów

Python - lista wbudowanych funkcji

A

abs()
aiter()
all()
anext()
any()
ascii()

B

bin()
bool()
breakpoint()
bytearray()
bytes()

C

callable()
chr()
classmethod()
compile()
complex()

D

delattr()
dict()
dir()
divmod()

E

enumerate()
eval()
exec()

F

filter()
float()
format()
frozenset()

G

getattr()
globals()

H

hasattr()
hash()
help()
hex()

I

id()
input()
int()
isinstance()
issubclass()
iter()

Python - lista wbudowanych funkcji

L

len()
list()
locals()

M

map()
max()
memoryview()
min()

N

next()

O

object()
oct()
open()
ord()

P

pow()
print()
property()

R

range()
repr()
reversed()
round()

S

set()
setattr()
slice()
sorted()
staticmethod()
str()
sum()
super()

T

tuple()
type()

V

vars()

Z

zip()

_

__import__()

Python - lista wbudowanych stałych

- True - reprezentuje wartość logiczną prawda (typ bool)
- False - reprezentuje wartość logiczną fałsz (typ bool)
- None - reprezentuje brak wartości lub wartość null, obiekt używany gdy np. domyślne argumenty nie są przekazywane do funkcji (typ NoneType)
- NotImplemented - specjalna wartość używana do wskazania, że metoda nie jest zaimplementowana dla określonych typów danych
- Ellipsis - reprezentowany przez ... (trzy kropki); używany głównie w specjalnych przypadkach, takich jak definiowanie zakresów lub wycinków w tablicach wielowymiarowych
- __debug__ - stała, która ma wartość True, jeśli Python działa w trybie debugowania
- copyright, credits, license - stałe informacyjne wyświetlające informacje o prawach autorskich, kredytach i licencji Pythona

Python - lista wbudowanych stałych

```
print(copyright)
print(credits)
print(license)
```

```
Copyright (c) 2001-2023 Python Software Foundation.
All Rights Reserved.

Copyright (c) 2000 BeOpen.com.
All Rights Reserved.

Copyright (c) 1995-2001 Corporation for National Research
Initiatives.
All Rights Reserved.

Copyright (c) 1991-1995 Stichting Mathematisch Centrum,
Amsterdam.
All Rights Reserved.
    Thanks to CWI, CNRI, BeOpen.com, Zope Corporation and a cast
of thousands
    for supporting Python development.  See www.python.org for
more information.
Type license() to see the full license text
```

Python - lista wbudowanych typów

- typy liczbowe:
  - int - liczby całkowite, dowolnie duże (bez limitu wielkości) , np. a = 23
  - float - liczby zmiennoprzecinkowe (rzeczywiste), jak double w C, np. b = 2.5
  - complex - liczby zespolone z częścią rzeczywistą i urojoną, np. z = 2 + 5j
- typy sekwencyjne:
  - list - lista, dynamiczna tablica elementów, które mogą być różnych typów, mutowalna - można zmieniać jej zawartość po utworzeniu, np. lst = [1, 2, 3]
  - tuple - krotka, niezmienna sekwencja elementów, po utworzeniu nie można zmieniać jej zawartości, np. kr = (1, 2, 3, 'x', 'y', 'z')
  - range - generuje sekwencję liczb całkowitych w sposób leniwy (liczby generowane są na bieżąco, a nie przechowywane w pamięci), np. z = range(100)
- typy tekstowe:
  - str - łańcuch znaków, czyli tekst; kodowanie znaków - Unicode, np. txt = "Witaj"

Python - lista wbudowanych typów

- typy binarne:
  - bytes - sekwencja bajtów (wartości od 0 do 255); typ niemutowalny - po utworzeniu nie można zmienić jego zawartości, np. b = b'witaj'
  - bytearray - sekwencja bajtów; typ mutowalny - można zmienić zawartość, np. ba = bytearray(b'witaj')
  - memoryview - umożliwia efektywne operacje na dużych obiektach binarnych bez potrzeby ich kopiowania, np. mv = memoryview(b'witaj')
- typy zbiorów (zestawów):
  - set - modyfikowalny zbiór unikalnych elementów, np. s = {1, 2}
  - frozenset - niemodyfikowalny zbiór unikalnych elementów, np. fs = frozenset([1, 2, 3, 'x', 'y', 'z'])
- typy mapowania:
  - dict - słownik, kolekcja par klucz-wartość, klucze są unikalne, każdemu kluczowi przypisana jest jedna wartość, np. d = {'k1': 'w1', 'k2': 'w2'}

Python - lista wbudowanych typów

- typy logiczne:
  - bool - reprezentuje wartości logiczne True i False, podtyp typu int (dziedziczy wszystkie jego właściwości), np. b = True
- typy specjalne:
  - NoneType - reprezentowany tylko przez jeden obiekt None, oznacza brak wartości lub nieistnienie, np. n = None
- typy wbudowane obsługujące iterację:
  - enumerate - funkcja zwracająca iterator, który produkuje krotki zawierające indeks i element z sekwencji, np. e = enumerate(['x', 'y', 'z'])
  - reversed - funkcja zwracająca iterator odwracający kolejność elementów, np. r = reversed([6, 7, 8])
  - zip - funkcja zwracająca iterator, który łączy elementy z kilku iterowalnych obiektów. np. z = zip([6, 7, 8], ['x', 'y', 'z'])

Python - lista wbudowanych typów

- typy funkcyjne:
  - function - funkcja zdefiniowana za pomocą słowa kluczowego def lub lambda, np. def funkcja(x): return x ** 2
  - lambda - funkcja anonimowa, z dowolną liczbą argumentów, ale tylko jednym wyrażeniem, np. f = lambda x: x ** 2
- typy klas i obiektów:
  - class - klasy zdefiniowane za pomocą słowa kluczowego class, np. class MyClass: pass
  - object - podstawowa klasa bazowa dla wszystkich klas, np. o = object()
- typy wyjątków
  - BaseException - podstawowa klasa dla wszystkich wyjątków
  - Exception - podstawowa klasa dla większości wyjątków, używanych w typowych sytuacjach błędów programistycznych

Python - lista wbudowanych wyjątków

- wyjątki (ang. exceptions) są to specjalne obiekty, którymi posługuje się Python podczas zarządzania błędami, które mogą pojawić się w trakcie wykonywania programu

```
try:
    # kod, który może generować wyjątek
except ExceptionType:
    # obsługa wyjątku
finally:
    # kod, który zostanie wykonany zawsze
```

```
try:
    # kod, który może generować wyjątek
except ExceptionType:
    # obsługa wyjątku
else:
    # kod, który zostanie wykonany tylko wtedy,
    # gdy nie wystąpił żaden wyjątek
```

Python - lista wbudowanych wyjątków

- Exception - podstawowa klasa wszystkich wyjątków w Pythonie, wbudowane wyjątki są pochodnymi tej klasy.
- BaseException - podstawowa klasa wszystkich wyjątków, w tym takich, które nie są błędami (nie należy używać jej bezpośrednio)
- KeyboardInterrupt - występuje, gdy użytkownik przerywa wykonywanie programu za pomocą skrótu klawiaturowego (Ctrl+C)
- SystemExit - występuje, gdy program kończy działanie poprzez wywołanie funkcji sys.exit(), przechwycenie zapobiega zamknięciu programu
- StopIteration - wywoływany przez iteratory, aby wskazać koniec iteracji (automatycznie obsługiwany przez pętle for)
- AttributeError - wywoływany przy próbie dostępu do nieistniejącego atrybutu obiektu
- EOFError - wywoływany, gdy funkcje wczytujące dane z wejścia (np. input() lub read()) napotkają koniec pliku (EOF)

Python - lista wbudowanych wyjątków

- ArithmeticError - klasa podstawowa dla wyjątków związanych z błędami arytmetycznymi:
  - ZeroDivisionError - występuje przy próbie dzielenia liczby przez zero
  - OverflowError - występuje, gdy wynik operacji arytmetycznej jest zbyt duży, aby można go było reprezentować
  - FloatingPointError - występuje przy błędach związanych z operacjami zmiennoprzecinkowymi
- ImportError - występuje, gdy nie uda się zaimportować modułu
  - ModuleNotFoundError - występuje, gdy moduł nie może zostać odnaleziony
- IndexError - występuje, gdy indeks sekwencji (np. listy) jest poza zakresem
- KeyError - występuje, gdy klucz nie zostanie odnaleziony w słowniku
- MemoryError - występuje, gdy nie ma wystarczającej ilości pamięci, aby zakończyć operację

Python - lista wbudowanych wyjątków

- NameError - występuje, gdy zmienna lub symbol nie zostały zdefiniowane
- OSError - podstawowa klasa wyjątków związanych z błędami systemowymi
  - FileNotFoundError - plik lub katalog nie mogą zostać odnalezione
  - PermissionError - brak uprawnień do wykonania operacji
  - IsADirectoryError - operacja wymaga pliku, ale obiekt jest katalogiem
  - NotADirectoryError - operacja wymaga katalogu, ale obiekt nie jest katalogiem
- TypeError - występuje, gdy operacja lub funkcja jest wykonywana na obiekcie niewłaściwego typu
- ValueError - występuje, gdy funkcja otrzymuje argument o poprawnym typie, ale o niewłaściwej wartości

Python - moduł string

- moduł string zawiera różne funkcje i stałe, przydatne podczas wykonywania operacji na łańcuchach znaków
- lista stałych:
  - string.ascii_letters - zawiera wszystkie litery alfabetu (małe i duże)

```
import string
print(string.ascii_letters)
```

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
```

  - string.ascii_lowercase - zawiera wszystkie małe litery alfabetu
  - string.ascii_uppercase - zawiera wszystkie wielkie litery alfabetu
  - string.digits - zawiera wszystkie cyfry dziesiętne (0-9)
  - string.hexdigits - zawiera wszystkie cyfry szesnastkowe (0-9, a-f, A-F)
  - string.octdigits - zawiera wszystkie cyfry ósemkowe (0-7)

Python - moduł string

- lista stałych:
  - string.punctuation - zawiera wszystkie znaki interpunkcyjne
  - string.printable - zawiera wszystkie znaki, które są uznawane za "drukowalne", czyli litery, cyfry, znaki interpunkcyjne i białe znaki
  - string.whitespace - zawiera białe znaki (spacje, tabulatory, nowe linie itp.)

```
import string
txt = """W lutym 2024 r. powstalo 11 567 sztuk nowych
instalacji pv. Ich laczna moc wyniosla 232,50 MW.
Najwieksza grupe stanowia mikroinstalacje: 11 478
sztuk o lacznej mocy 100,33 MW."""

cyfry = [zn for zn in txt if zn in string.digits]
print(f"Liczba cyfr w tekście: {len(cyfry)}")
```

```
Liczba cyfr w tekście: 24
```

Python - moduł datetime

- moduł datetime jest używany do manipulowania datami i czasem, zawiera wiele funkcji i klas, które ułatwiają operacje na datach, takie jak tworzenie, porównywanie, formatowanie i wykonywanie operacji arytmetycznych
- klasy znajdujące się w module:
  - datetime - reprezentuje datę i czas jako pojedynczy obiekt
  - date - reprezentuje tylko datę (bez czasu)
  - time - reprezentuje tylko czas (bez daty)
  - timedelta - reprezentuje różnicę pomiędzy dwoma datami lub czasami
- klasy umożliwiają wykonywanie operacji arytmetycznych na datach i czasach, takich jak dodawanie i odejmowanie
- obiekty klas mogą być porównywane ze sobą za pomocą operatorów porównania, takich jak <, <=, >, >=, ==, !=
- obiekty posiadają metody umożliwiające dostęp do składowych daty i czasu, takich jak rok, miesiąc, dzień, godzina, minuta, sekunda

Python - moduł datetime

```
import datetime

dowolna_data = datetime.datetime(2024, 5, 1, 12, 0, 0)
biezaca_data = datetime.datetime.now()

print("Dowolna data:")
print("Rok:    ", dowolna_data.year)
print("Miesiąc:", dowolna_data.month)
print("Dzień:  ", dowolna_data.day)

print("\nTeraz jest:")
print("Rok:    ", biezaca_data.year)
print("Miesiąc:", biezaca_data.month)
print("Dzień:  ", biezaca_data.day)
print("Godzina:", biezaca_data.hour)
print("Minuta: ", biezaca_data.minute)
print("Sekunda:", biezaca_data.second)
```

```
Dowolna data:
Rok:     2024
Miesiąc: 5
Dzień:   1

Teraz jest:
Rok:     2025
Miesiąc: 5
Dzień:   13
Godzina: 9
Minuta:  9
Sekunda: 22
```

Python - moduł zoneinfo

- klasa zoneinfo umożliwia obsługę stref czasowych (dostępna od wersji 3.9)
- pozwala tworzyć obiekty reprezentujące konkretne strefy czasowe, takie jak "Europe/Warsaw", "America/New_York", "Asia/Tokyo", itp.
- obiekty zoneinfo zawierają informacje o przesunięciach czasowych, historii zmian czasu letniego/zimowego i innych ustawieniach specyficznych dla danej strefy czasowej
- klasa zoneinfo umożliwia wykonywanie operacji na strefach czasowych, takich jak konwersja daty i czasu między różnymi strefami czasowymi, określanie czy dana data i czas należy do określonej strefy czasowej, itp.
- obiekty zoneinfo automatycznie uwzględniają zmiany czasu letniego/zimowego

Python - moduł zoneinfo

```
from datetime import datetime
from zoneinfo import ZoneInfo

warsaw_time = datetime(2025, 5, 13, 10, 15,
                       tzinfo=ZoneInfo("Europe/Warsaw"))
print("Czas w Warszawie:  ", warsaw_time)

ny_time = warsaw_time.astimezone(ZoneInfo("America/New_York"))
print("Czas w Nowym Jorku:", ny_time)

tokyo_time = warsaw_time.astimezone(ZoneInfo("Asia/Tokyo"))
print("Czas w Tokyo:      ", tokyo_time)
```

```
Czas w Warszawie:   2025-05-13 10:15:00+02:00
Czas w Nowym Jorku: 2025-05-13 04:15:00-04:00
Czas w Tokyo:       2025-05-13 17:15:00+09:00
```

- poprawne działanie programu wymaga zainstalowania pakietu tzdata
  pip install tzdata

Python - moduł calendar

- moduł calendar umożliwia generowanie kalendarzy oraz wykonywanie operacji związanych z datami
- pozwala na tworzenie kalendarzy dla różnych lat i miesięcy, dostęp do informacji o dniach tygodnia, daty, a także wykonywanie różnych operacji arytmetycznych na datach
- moduł calendar zawiera funkcje do generowania kalendarzy dla określonego roku i miesiąca, takie jak calendar.month() i calendar.calendar()
- klasa TextCalendar umożliwia generowanie kalendarzy w postaci tekstu, można dostosować sposób wyświetlania kalendarza, takie jak wybór pierwszego dnia tygodnia i formatowanie tekstu
- moduł zawiera funkcje pomocnicze, np. calendar.weekday() - zwraca dzień tygodnia dla określonej daty, calendar.monthrange() - zwraca pierwszy dzień tygodnia oraz liczbę dni w danym miesiącu
- moduł calendar zawiera również stałe, takie jak calendar.MONDAY, calendar.TUESDAY, itd., które reprezentują dni tygodnia

Python - moduł calendar

```
import calendar

print("Kalendarz miesiąca maj 2025:")
print(calendar.month(2025, 5))
```

```
Kalendarz miesiąca maj 2025:
      May 2025
Mo Tu We Th Fr Sa Su
            1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
```

Python - moduł calendar

```
import calendar

print("Numer dnia tygodnia dla 16 maja 2025 roku:")
print(calendar.weekday(2025, 5, 16))

print("Początkowy dzień tygodnia i liczba dni w styczniu
2025 roku:")
print(calendar.monthrange(2025, 1))

print("Stałe reprezentujące dni tygodnia:")
print("Poniedziałek:", calendar.MONDAY)
print("Wtorek:", calendar.TUESDAY)
```

```
Numer dnia tygodnia dla 16 maja 2025 roku:
4
Początkowy dzień tygodnia i liczba dni w styczniu 2025 roku:
(calendar.WEDNESDAY, 31)
Stałe reprezentujące dni tygodnia:
Poniedziałek: 0
Wtorek: 1
```

Python - moduł calendar

```
import calendar

print(calendar.calendar(2025))
```

```
                                  2025

      January                   February                  March
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
       1  2  3  4  5                  1  2                     1  2
 6  7  8  9 10 11 12       3  4  5  6  7  8  9       3  4  5  6  7  8  9
13 14 15 16 17 18 19      10 11 12 13 14 15 16      10 11 12 13 14 15 16
20 21 22 23 24 25 26      17 18 19 20 21 22 23      17 18 19 20 21 22 23
27 28 29 30 31            24 25 26 27 28            24 25 26 27 28 29 30
                                                    31

      April                     May                       June
Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su      Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6                  1  2  3  4                        1
 7  8  9 10 11 12 13       5  6  7  8  9 10 11       2  3  4  5  6  7  8
14 15 16 17 18 19 20      12 13 14 15 16 17 18       9 10 11 12 13 14 15
21 22 23 24 25 26 27      19 20 21 22 23 24 25      16 17 18 19 20 21 22
28 29 30                  26 27 28 29 30 31         23 24 25 26 27 28 29
                                                    30
```

Koniec wykładu nr 6

Dziękuję za uwagę!