Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny

Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia

Rok akademicki 2023/2024

Wykład nr 10 (15.05.2024)

dr inż. Jarosław Forenc

Plan wykładu nr 10

- Pliki w Pythonie - moduł pathlib
  - odczyt z pliku tekstowego - metoda read_text()
  - zapis do pliku tekstowego - metoda write_text()
  - inne metody
- Wyjątki
  - try-except-finally
  - try-except-else
- Programowanie obiektowe
  - definicje
  - struktura klasy
  - konstruktor
  - atrybuty i metody

Python - pliki (moduł pathlib)

- moduł pathlib używany jest do obsługi ścieżek plików i katalogów (niezależnej od systemu operacyjnego)
- ścieżki reprezentowane są jako obiekty Path, które można manipulować za pomocą różnych metod
- metody umożliwiają także odczyt danych z pliku i ich zapis do pliku
- odczytanie i wyświetlenie zawartości pliku tekstowego na ekranie:

```
from pathlib import Path
plik = Path("dane.txt")
dane = plik.read_text()
print(dane)
```

- tworzymy obiekt Path i przypisujemy go do zmiennej plik
- metoda read_text() wczytuje całą zawartość pliku i umieszcza w pojedynczym, długim ciągu tekstowym o nazwie dane, a następnie automatycznie zamyka plik

Python - pliki (moduł pathlib)

- po dotarciu do końca pliku, metoda read_text() zwraca pusty ciąg tekstowy, który jest wyświetlany w postaci pustego wiersza
- pusty wiersz można usunąć metodą rstrip():

```
from pathlib import Path
plik = Path("dane.txt")
dane = plik.read_text()
dane = dane.rstrip()
print(dane)
```

- można zastosować także tzw. łączenie metod:

```
from pathlib import Path
plik = Path("dane.txt")
dane = plik.read_text().rstrip()
print(dane)
```

Python - pliki (moduł pathlib)

- jako ścieżkę dostępu można podać:
  - ścieżkę względną - lokalizacja pliku jest określana względem katalogu, w którym znajduje się wykonywany program

```
plik = Path("folder/podfolder/plik.txt")
```

  - ścieżkę bezwzględną - zawiera pełną ścieżkę dostępu do pliku, rozpoczyna się od nazwy dysku (Windows) lub korzenia systemu plików (Linux)

```
plik1 = Path("C:/folder/podfolder/plik.txt")
plik2 = Path("/home/user/folder/plik.txt")
```

- przy zapisie ścieżek dostępu podajemy znaki ukośnika (/) do oddzielenia poszczególnych ich elementów (np. katalogów)

Python - pliki (moduł pathlib)

- tekst odczytany funkcją read_text() można podzielić na wiersze, a następnie przetwarzać przy wykorzystaniu pętli for

```
from pathlib import Path
plik = Path("dane.txt")
dane = plik.read_text()
wiersze = dane.splitlines()
for wiersz in wiersze:
    print(wiersz)
```

- metoda splitlines() dzieli łańcuch znaków na linie, domyślnie znakiem podziału jest znak nowego wiersza (\n)
- podział na linie można umieścić bezpośrednio w pętli for

```
...
dane = plik.read_text()
for wiersz in dane.splitlines():
    print(wiersz)
```

Python - pliki (moduł pathlib)

- jeśli chcemy na danych z pliku pracować jak na liczbach, to musimy skonwertować je z tekstu do liczby całkowitej (funkcja int()) lub zmiennoprzecinkowej (funkcja float())
- suma liczb zmiennoprzecinkowych znajdujących się w pliku

```
from pathlib import Path
plik = Path("liczby.txt")
wiersze = plik.read_text().splitlines()
suma = 0
for wiersz in wiersze:
    suma = suma + float(wiersz)
print(f"Suma liczb w pliku: {suma}")
```

```
12.34
15.67
21.36
45.12
```

```
Suma liczb w pliku: 94.49
```

- Python nie ma narzuconego ograniczenia wskazującego maksymalną ilość danych, z jaką może pracować - ograniczeniem jest pamięć w systemie

Python - pliki (moduł pathlib)

- metoda write_text() umożliwia zapis jednego wiersza tekstu do pliku

```
from pathlib import Path
plik = Path("zapis.txt")
plik.write_text("Witaj świecie!\n")
```

```
Witaj świecie!
```

- jeśli pliku o podanej nazwie nie ma, to zostanie utworzony
- jeśli plik o podanej nazwie istnieje, to jego poprzednia zawartość jest usuwana
- metoda write_text() gwarantuje poprawne zamknięcie pliku po przeprowadzeniu operacji zapisu

Python - pliki (moduł pathlib)

- jeśli tekst składa się z większej liczby wierszy, to należy go wcześniej przygotować i wywołać metodę write_text() tylko raz

```
from pathlib import Path
tekst =  "-------------------------\n"
tekst += "| Nazwa |  Kod  |  Kurs  |\n"
tekst += "-------------------------\n"
tekst += "| euro  | 1 EUR | 4.2812 |\n"
tekst += "| dolar | 1 USD | 3.9441 |\n"
tekst += "-------------------------\n"
plik = Path("tabela.txt")
plik.write_text(tekst)
```

```
-------------------------
| Nazwa |  Kod  |  Kurs  |
-------------------------
| euro  | 1 EUR | 4.2812 |
| dolar | 1 USD | 3.9441 |
-------------------------
```

Python - pliki (metody modułu pathlib)

Metoda	Opis
`cwd()`	zwraca obiekt Path reprezentujący bieżący katalog
`home()`	zwraca obiekt Path reprezentujący katalog domowy użytkownika
`exists()`	zwraca wartość True, gdy dana ścieżka pliku lub katalogu istnieje fizycznie na dysku
`is_dir()`	zwraca wartość True, gdy dana ścieżka reprezentuje katalog
`is_file()`	zwraca wartość True, gdy dana ścieżka reprezentuje plik
`iterdir()`	iteruje przez wszystkie elementy (pliki, katalogi, itp.) w danym katalogu, zwracając generator zawierający obiekty Path reprezentujące te elementy

Python - pliki (metody modułu pathlib)

Metoda	Opis
`mkdir()`	tworzy nowy katalog na dysku
`read_bytes()`	odczytuje zawartość pliku w postaci danych binarnych
`rename()`	zmiana nazwy pliku lub katalogu
`replace()`	zastępuje plik lub katalogu na dysku; podobna do metody rename(), ale jeśli docelowa ścieżka już istnieje, zostanie zastąpiona przez aktualny plik lub katalog
`rmdir()`	usuwa pusty katalog z systemu plików
`write_bytes()`	zapisuje dane do pliku w postaci binarnej

Python - pliki (moduł pathlib)

- sprawdzenie czy plik istnieje, wyświetlenie zawartości pliku, wyświetlenie bieżącego i domowego katalogu

```
from pathlib import Path
plik = Path("dane.txt")
if plik.exists():
    if plik.is_file():
        print(f"{plik} istnieje, zawartość: ")
        dane = plik.read_text()
        print(dane)
    else:
        print(f"{plik} - to nie jest plik")
else:
    print(f"Brak pliku: {plik}")

print(f"Bieżący katalog: {Path.cwd()}")
print(f"Katalog domowy: {Path.home()}")
```

Python - wyjątki

- wyjątki (ang. exceptions) są to specjalne obiekty, którymi posługuje się Python podczas zarządzania błędami, które mogą pojawić się w trakcie wykonywania programu
- brak obsługi zgłoszonego wyjątku przerywa program i powoduje wyświetlenie stosu wywołań informującego o zgłoszonym wyjątku

```
x = float(input("Podaj x: "))
y = float(input("Podaj y: "))
z = x / y
print(f"Wynik to: {z}")
```

```
Podaj x: 3
Podaj y: 0
Traceback (most recent call last):
  File "d:\MyApp.py", line 3, in <module>
    z = x / y
        ~~^~~
ZeroDivisionError: float division by zero
```

Python - wyjątki (try-except-finally)

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

- instrukcja try-except-finally jest używana, gdy chcemy mieć pewność, że pewne instrukcje zostaną wykonane niezależnie od tego, czy wyjątek wystąpi, czy nie
- blok finally jest opcjonalny, ale jeśli jest obecny, zostanie wykonany zawsze, niezależnie od tego, czy wystąpił wyjątek, czy nie
- blok finally jest używany do czyszczenia zasobów, takich jak pliki, połączenia sieciowe itp., które powinny być zawsze zwalniane niezależnie od tego, czy wystąpił wyjątek

Python - wyjątki (try-except-else)

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

- instrukcja try-except-else jest używana, gdy chcemy wykonać pewne instrukcje tylko wtedy, gdy nie wystąpił żaden wyjątek w bloku try
- blok else jest opcjonalny i zostanie wykonany tylko wtedy, gdy w bloku try nie wystąpi żaden wyjątek
- blok else jest przydatny, gdy chcemy wykonać pewne operacje, np. wykonać pewne obliczenia na danych, które są oczekiwane na poprawne działanie

Python - wyjątki

- zabezpieczenie programu przed dzieleniem przez zero

```
x = float(input("Podaj x: "))
y = float(input("Podaj y: "))
try:
    z = x / y
except ZeroDivisionError:
    print("Błąd dzielenia przez zero!")
else:
    print(f"Wynik to: {z}")
```

```
Podaj x: 3
Podaj y: 0
Błąd dzielenia przez zero!
```

```
Podaj x: 3
Podaj y: 7
Wynik to: 0.42857142857142855
```

Python - wyjątki

- zabezpieczenie programu przed dzieleniem przez zero i błędnym wprowadzeniem danych

```
try:
    x = float(input("Podaj x: "))
    y = float(input("Podaj y: "))
    z = x / y
except ZeroDivisionError:
    print("Błąd dzielenia przez zero!")
except ValueError:
    print("Błędny zapis liczby!")
else:
    print(f"Wynik to: {z}")
```

```
Podaj x: 3
Podaj y: 7,0
Błędny zapis liczby!
```

Python - wyjątki

- statystyka liter w pliku tekstowym

```
try:
    with open("tadeusz.txt", "r", encoding="utf-8") as plik:
        dane = plik.read()
except FileNotFoundError:
    print("Brak pliku tadeusz.txt")
else:
    statystyka= {}
    for znak in dane:
        if znak.isalpha():
            statystyka[znak] = statystyka.get(znak, 0) + 1
    print("Statystyka liter:")
    for litera, liczba in sorted(statystyka.items()):
        print(f"{litera}: {liczba}")
```

- metoda get() zwraca wartość dla klucza znak lub wartość domyślną (0) jeśli klucz nie istnieje w słowniku

Python - wyjątki

- statystyka liter w pliku tekstowym

```
Litwo! Ojczyzno moja! ty jesteś jak zdrowie:
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
```

```
D: 1
I: 1
K: 1
L: 1
O: 1
W: 1
a: 5
b: 4
c: 6
d: 4
```

```
e: 11
i: 16
j: 6
k: 4
l: 2
m: 1
n: 5
o: 14
p: 3
r: 3
```

```
s: 5
t: 11
u: 1
w: 5
y: 3
z: 7
ą: 1
ć: 2
ę: 8
ł: 2
ś: 3
```

Python - wyjątki

- jeśli po wystąpieniu wyjątku nie chcemy nic robić, ale chcemy go obsłużyć, to wtedy po except wpisujemy polecenie pass

```
try:
    x = float(input("Podaj x: "))
    y = float(input("Podaj y: "))
    z = x / y
except ZeroDivisionError:
    print("Błąd dzielenia przez zero!")
except ValueError:
    pass
else:
    print(f"Wynik to: {z}")
```

- polecenie pass zazwyczaj stosuje się wtedy, gdy na razie nie wpisujemy żadnego kodu, ale planujemy dopisać go w przyszłości

Python - programowanie obiektowe (definicje)

- programowanie obiektowe to paradygmat programowania, w którym programy są konstruowane poprzez definiowanie i manipulowanie obiektami
- w Pythonie programowanie obiektowe opiera się na trzech głównych elementach: klasach, obiektach i dziedziczeniu
- klasa to szablon lub wzorzec, który definiuje cechy i zachowania obiektów (w pewnym sensie tworzy nowy typ)
- obiekt jest instancją (egzemplarzem) klasy (zmienną utworzonego typu)
- atrybuty to cechy obiektów, które przechowują dane; w Pythonie można dodać nowe atrybuty do obiektu w dowolnym momencie
- metody to funkcje zdefiniowane wewnątrz klasy, które działają na obiektach danej klasy; metody te mają dostęp do atrybutów obiektu i mogą nimi manipulować
- dziedziczenie umożliwia tworzenie nowych klas (pochodnych) na podstawie istniejących klas (bazowych lub nadrzędnych); klasa pochodna dziedziczy atrybuty i metody klasy bazowej

Python - programowanie obiektowe (klasy)

- ogólna struktura definicji klasy w Pythonie

```
class NazwaKlasy:
    # Atrybuty klasy
    zmienna = "wartość"

    # Konstruktor
    def __init__(self, parametry):
        self.parametry = parametry

    # Metody klasy
    def metoda(self):
        # kod metody
        return something
```

- klasa składa się z nagłówka klasy oraz ciała klasy
- nagłówek rozpoczyna się od słowa kluczowego class, po którym podaje nazwę klasy
- przyjęło się, że nazwa klasy rozpoczyna się wielką literą
- ciało klasy zawiera atrybuty i metody klasy, które definiują zachowanie i cechy obiektów tworzonych na podstawie tej klasy

Python - programowanie obiektowe (klasy)

- definicja klasy Osoba, deklaracja obiektu Nowak

```
class Osoba:

    def __init__(self, imie, nazwisko, wiek):
        self.imie = imie
        self.nazw = nazwisko
        self.wiek = wiek

    def drukuj(self):
        print(f"{self.imie} {self.nazw}, {self.wiek} lat")

nowak = Osoba("Jan","Nowak",25)
nowak.drukuj()
```

```
Jan Nowak, 25 lat
```

Python - programowanie obiektowe (konstruktor)

```
def __init__(self, imie, nazwisko, wiek):
    self.imie = imie
    self.nazw = nazwisko
    self.wiek = wiek
```

- __init__ - konstruktor, specjalna metoda wywoływana podczas tworzenia nowej instancji klasy (nowego obiektu)

```
nowak = Osoba("Jan","Nowak",25)
```

- dwa znaki podkreślenia na początku i końcu to konwencja zapisu stosowana w celu uniknięcia sytuacji kolizji z innymi metodami o takich samych nazwach, które mogą być zdefiniowane później
- pierwszy parametr konstruktora powinien nazywać się self - jest to odwołanie do danego egzemplarza obiektu klasy
- dzięki self egzemplarze obiektów klasy uzyskują dostęp do atrybutów i metod zdefiniowanych w klasie

Python - programowanie obiektowe (konstruktor)

```
def __init__(self, imie, nazwisko, wiek):
    self.imie = imie
    self.nazw = nazwisko
    self.wiek = wiek
```

- __init__ - konstruktor, specjalna metoda wywoływana podczas tworzenia nowej instancji klasy (nowego obiektu)

```
nowak = Osoba("Jan","Nowak",25)
```

- do konstruktora przekazywane są argumenty imie, nazwisko i wiek, zaś argument self jest przekazywany automatycznie
- zmienne poprzedzone self to atrybuty i są one dostępne dla każdej metody w klasie

Python - programowanie obiektowe (konstruktor)

```
def __init__(self, imie, nazwisko, wiek):
    self.imie = imie
    self.nazw = nazwisko
    self.wiek = wiek
```

- __init__ - konstruktor, specjalna metoda wywoływana podczas tworzenia nowej instancji klasy (nowego obiektu)

```
nowak = Osoba("Jan","Nowak",25)
```

- self.imie = imie pobiera wartość przechowywaną w parametrze imie, umieszcza ją w atrybucie imie, który następnie zostaje dołączony do tworzonego egzemplarza klasy
- w konstruktorze nie występuje return, ale Python automatycznie zwraca egzemplarz klasy Osoba

Python - programowanie obiektowe (metody)

```
def drukuj(self):
    print(f"{self.imie} {self.nazw}, {self.wiek} lat")
```

- metoda drukuj() nie wymaga innych danych więc ma tylko jeden parametr, czyli self
- wywołanie metody:

```
nazwa_egzemplarza.nazwa_metody(argumenty)
```

- wywołanie metody drukuj() na rzecz egzemplarza obiektu nowak:

```
nowak = Osoba("Jan","Nowak",25)
nowak.drukuj()
```

Python - programowanie obiektowe (__str__)

- zamiast metody drukuj() można zastosować wbudowaną metodę __str__, która jest wywoływana automatycznie, gdy obiekt jest konwertowany na napis za pomocą funkcji str() lub gdy obiekt jest używany w kontekście, gdzie oczekiwany jest napis, na przykład w funkcji print()

```
class Osoba:

    def __init__(self, imie, nazwisko, wiek):
        self.imie = imie
        self.nazw = nazwisko
        self.wiek = wiek

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

nowak = Osoba("Jan","Nowak",25)
print(nowak)
```

```
Jan Nowak, 25 lat
```

Koniec wykładu nr 10

Dziękuję za uwagę!