Operacje na plikach

Opercaje na plikach można podzielić na trzy etapy:

1. Otwarcie pliku
2. Działanie na pliku (odczyt lub zapis)
3. Zamknięcie pliku

plik = open(nazwa, tryb, bufor)
gdzie:

plik – nazwa obiektu, którą sami nadajemy
nazwa – nazwa pliku na dysku, jaka jest
tryb – tryb otwarcia pliku (np. do odczytu, do zapisu itd.)
bufor – obszar pamięci przechowujący dane w oczekiwaniu na zapis i odczyt

Wybrane tryby otwarcia plików:
r – tylko do odczytu. Plik musi istnieć:
w – tylko do zapisu. Jeżeli pliku nie ma to zostanie utworzony a jeżeli jest to jego zawartość zostanie zastąpiona nową.
a – do dopisywania. Dane dopisują się na końcu pliku. Jeśli plik nie istnieje to zostanie utworzony
r+ - do odczytu i zapisu. Plik musi istnieć.
w+ - do odczytu i zapisu. Jeśli plik nie istnieje zostanie utworzony.
a+ - do odczytu i zapisu. Jeśli plik nie istnieje zostanie utworzony.

Do odczytania danych z pliku można użyć komend:

- read(rozmiar) – odczytuje dane o rozmiarze, jeśli podany
- readline(rozmiar) – odczytuje wiersze lub ilości znaków jeśli podano rozmiar
- readlines() – odczytuje wiersze z pliku

Przykład 1

```
plik = open("tekst.txt","r")
#odczyt 10 znaków
znaki = plik.read(10)
#odczyt jednej lini z pliku
linia = plik.readline()
#odczyt wierszy z pliku
wiersze = plik.readlines()
#zamknięcie pliku
plik.close()
#drukujemy 10 znaków
print(znaki)
print("\n")
#drukujemy linie
print(linia)
print("\n")
#drukujemy wiersze
print(wiersze)
```

Uwaga 1

Jeśli otwieramy plik i odczytujemy z niego dane jak wyżej to wskaźnik aktualnej pozycji w pliku się przemieszcza. Dlatego w wyniku najpierw otrzymamy pierwsze 10 znaków, potem następne znaki z pozostałej linijki a na koniec resztę linijek tekstu z pliku.

Uwaga 2

Po zakończeniu działania skryptu wszystkie otwarte pliki zamykane są automatycznie.

Do zapisywania danych do pliku możemy użyć:

- write(łańcuch) – zapisuje dane ze zmiennej łańcuch
- writelines(lista) – zapisuje dane z listy

Przykład 2

```
import sys
print("Podaj kierunek studiów, rok i specjalność")
#odczyt danych ze standardowego wyjścia
dane = sys.stdin.readline()
#otwarcie pliku
plik = open("dane.txt","w+")
#zapisanie do pliku
plik.write(dane)
#zamykamy plik
plik.close()
#tworzymy liste
lista = []
for x in range(6):
    lista += [x]
#otwarcie pliku do dopisania
plik = open("dane.txt","a+")
#zapisujemy
plik.writelines(str(lista))
#zamknięcie pliku
plik.close()
```

Przykład 3

Plik możemy otwierać do zapisu i odczytu za pomocą komendy with, wówczas nie musimy martwić się o zamknięcie pliku. Pętla for pozwala na wyświetlenie pliku linijka po linijce

```
with open("tekst.txt", "r") as plik:
    for linia in plik:
        print(linia,end="")
```

Programowanie obiektowe – wprowadzenie

Programowanie obiektowe – podstawowe terminy

Programowanie obiektowe wymaga zaprojektowania struktury opartej na danych i kodzie. Taka struktura nazywana jest klasą a każdy obiekt stworzony na podstawie tej klasy to instancja instancją (albo wystąpieniem) danej klasy. Dane powiązane z obiektem to będą atrybuty (inaczej własności lub właściwości) a funkcje, które wykonują coś na obiekcie to metody.

Enkapsulacja

Inaczej zwana hermetyzacją. Chodzi o to by tak zdefiniować klasę aby jej metody obsługiwały wszystkie właściwości obiektu i żeby żadna funkcja z zewnątrz nie mogła zmienić właściwości obiektu.

Dziedziczenie

Jeśli po utworzeniu klasy okaże się, że potrzebujemy podobnej. np. chcemy stworzyć szafę grającą, która dodatkowo odtwarza mp3 to możemy zastosować dziedziczenie. Wówczas nowa klasa otrzymuje wszystkie właściwości tej, po której dziedziczy. Oryginalną klasę nazywamy klasą bazową (inne nazwy to nadklasa lub superklasa) a nową klasą pochodną (lub podklasą).

Składnia:

```
class NazwaKlasy[(KlasaBazowa1, KlasaBazowa2, KlasaBazowa3)]:
  instrukcje1
  instrukcje2
  instrukcjeN
```

Przykład 4

Definicja pustej klasy i utworzenie obiektu

```
class PierwszaKlasa:
    """Pierwsza klasa python""" #składnia opisowa, opis tego
co jest umieszczone w klasie

obiekt = PierwszaKlasa()
print(obiekt)
```

Przykład 5

Dodajemy atrybut

```
class PierwszaKlasa:
    """Pierwsza klasa python"""
    atrybut = 54321

obiekt = PierwszaKlasa()
```

```
print(obiekt)
print(obiekt.atrybut)
```

Przykład 6

Dodajemy metodę

```
class PierwszaKlasa:
    """Przykład klasy"""
    atrybut = 54321
    def pierwsza_metoda(self):
        return "Teraz działa pierwsza Metoda"
obiekt = PierwszaKlasa()
print(obiekt)
print(obiekt.atrybut)
print(obiekt.pierwsza_metoda())
```

Przykład 7

Dodajemy atrybut do instancji klasy

```
class PierwszaKlasa:
    """Przykład klasy"""
    atrybut = 54321
    def pierwsza_metoda(self):
        return "Teraz działa pierwsza Metoda"
obiekt = PierwszaKlasa()
print(obiekt)
#drukujemy atrybut
print(obiekt.atrybut)
#drukujemy metodę
print(obiekt.pierwsza_metoda())
#dodajemy atrybut do istniejącego obiektu
obiekt.tekst = "la la la"
print(obiekt.tekst)
#ale go nie będzie w nowej instancji klasy
nowy_obiekt = PierwszaKlasa()
print(nowy_obiekt.tekst)
```

Konstruktory

Konstruktor to specjalna funkcja, która tworzy obiekt. Jeśli nie zdefiniujemy swojego konstruktora, to Python wywoła konstruktor domyślny. W Pythonie konstruktor nie tworzy instancji klasy a nadaje wartości początkowe do obiektu.

Przykład 8

```
class Ksztalty:
    # definicja konstruktora
    def __init__(self, x, y):
        #deklarujemy atrybuty
```

```
        #self wskazuje że chodzi o zmienne właśnie
definiowanej klasy
        self.x=x
        self.y=y
        self.opis = "To będzie klasa dla ogólnych kształtów"
    def pole(self):
        return self.x * self.y
    def obwod(self):
        return 2 * self.x + 2 * self.y
    def dodaj_opis(self, text):
        self.opis = text
    def skalowanie(self, czynnik):
        self.x = self.x * czynnik
        self.y = self.y * czynnik
# Tworzymy obiekt
kwadrat= Ksztalty(10,30)
# Sprawdzamy teraz jak działają metody które zwracają wartość
print(kwadrat.pole())
print(kwadrat.obwod())
# sprawdzamy jak działają metody, które nie zwracają wartości
kwadrat.dodaj_opis("Kwadrat")
print(kwadrat.opis)
kwadrat.skalowanie(0.5)
print(kwadrat.x)
print(kwadrat.y)
```

Uwaga 1

Self reprezentuje instancje klasy. Używając słowa kluczowego „self” możemy uzyskać dostęp do atrybutów i metod w klasie Python. Tworząc metody w klasie zawsze w parametrach używamy słowa kluczowego self.

Uwaga 2

Niektóre funkcje można poprzedzić znakami __ i dla nas będą miały specjalne znaczenie. Konwencja mówi, że wtedy będą to zmienne lub funkcje prywatne czyli takie, które są widoczne tylko dla jednej klasy i nie mogą być modyfikowane przez funkcje i zmienne z innej klasy. W rzeczywistości jest to tylko umowa bo w Pythonie nie ma prywatnych zmiennych czy funkcji, wszystkie są publiczne.

Przykład 9 Do uwaga 2.

```
class Ksztalty:
    # definicja zmiennej poprzedzonej __
    __jestem_prywatna__ = "xyz"

    # definicja konstruktora
    def __init__(self, x, y):
        # deklarujemy atrybuty
        # self wskazuje że chodzi o zmienne właśnie
definiowanej klasy
```

```
        self.x = x
        self.y = y
        self.opis = "To będzie klasa dla ogólnych kształtów"

    def pole(self):
        return self.x*self.y

    def obwod(self):
        return 2*self.x + 2*self.y

    def dodaj_opis(self, text):
        self.opis = text

    def skalowanie(self, czynnik):
        self.x = self.x * czynnik
        self.x = self.y * czynnik

    #Jakaś funkcja
    def zmieniam_tekst(self, tekst):
        tekst += " to jest to ;)”
        return tekst


# Tworzymy obiekt
kwadrat = Ksztalty(10,30)

# Sprawdzmy dostęp do zmiennej prywatnej
print(kwadrat.__jestem_prywatna__)

# a może uda nam się jeszcze zmienić wartość?
kwadrat.__jestem_prywatna__="na na na"
print(kwadrat.__jestem_prywatna__)

# spróbujmy czy nowa funkcja coś może zmienić
print(kwadrat.zmieniam_tekst(kwadrat.__jestem_prywatna__))
```