

Dokumentacja wstępna

Interpreter języka

Kacper Murygin, 318700

Celem projektu jest stworzenie języka, oferującego podstawowe własności języków programowania wraz z wbudowanym typem słownika.

Dostępne operacje na słowniku:

- dodawanie elementów
- usuwanie elementów
- wyszukiwanie elementów według klucza
- iterowanie po elementach zgodnie z zadaną kolejnością
- wykonanie na słowniku zapytań w stylu LINQ

1. Wymagania funkcjonalne i нефunkcjonalne:

- a) Program musi zawierać funkcję main ze zwracanym typem int
- b) Każda linia kodu kończy się średnikiem- „;”
- c) Język pozwala na pisanie komentarzy
- d) Można definiować własne funkcje, wywoływać je rekurencyjnie

2. Obsługa podstawowych typów danych liczbowych:

- a) Rodzaje danych liczbowych:
 - typ całkowitoliczbowy ze znakiem **int**
 - typ zmiennoprzecinkowy **float**
- b) Operacje matematyczne:
 - Mnożenie *
 - Dzielenie /
 - Dodawanie +
 - Odejmowanie -
 - Priorytet wykonywania operacji matematycznych-standardowy:
 - 1) wykonywanie działań wewnątrz nawiasów
 - 2) mnożenie i dzielenie
 - 3) dodawanie i odejmowanie

3. Obsługa typu znakowego:

- a) Typ znakowy **string**
- b) Dozwolone operacje- konkatencja
- c) String może zawierać dowolne znaki, w tym:
 - Wyróżnik stringa- cudzysłów
 - Znak nowej linii
 - Znak tabulacji

4. Konwersja typów

W języku będą dostępne dwa rodzaje konwersji typów:

- z int na float
- z float na int.

Pierwszy przypadek jest prostszy, do liczby całkowitej zostaje dodana część ułamkowa równa 0. W drugim przypadku zaś, część ułamkowa zostaje wycięta, pozostaje jedynie część całkowita.

```
int x = 10;  
float y = float(x);    // y = 10.0
```

```
float a = 5.9;  
int b = int(a);  // b = 5
```

W przypadku innych konwersji- niedozwolonych np. próba konwersji stringa na int, zostanie zgłoszony wyjątek.

5. Typ słownika Dict

- a) Dostępne atrybuty:
 - length- rozmiar słownika (ilość par klucz-wartość)
- b) Stworzenie nowej zmiennej o typie Dict- pusty słownik bez zawartości:
 - Dict<typ_klucza, typ_wartosci> nazwa_zmiennej = new Dict();
- c) Konstruktor słownika ze startową zawartością:
 - Dict<typ_klucza, typ_wartosci> nazwa_zm = new Dict(zawartość);

Przykłady:

```
Dict<string, int> dict_1 = new Dict();
```

```
Dict<string, int> dict_data = new Dict({"dzien": 1, "miesiac": 1, "rok": 2024});
```

- d) Dodawanie elementu do słownika:
 - nazwa_zmiennej[klucz] = wartosc;

```
dict_1["rok"] = 2024;
```

- e) Usuwanie elementu ze słownika – usuwany element o podanym kluczu:
 - nazwa_zmiennej.delete(klucz);

```
dict_1.delete("rok");
```

- f) Wyszukiwanie elementu według klucza:
 - nazwa_zmiennej.get(wartosc_klucza);

```
dict_data.get("rok");
```

- nazwa_zmiennej[klucz];

```
dict_data["rok"];
```

- g) Sprawdzanie, czy dany klucz znajduje się w słowniku:
 - nazwa_zmiennej.contains(wartosc_klucza);

```
dict_data.contains("dzien"); //zwróci true
```

```
dict_data.contains("naleśniki"); //zwróci false
```

h) Wykonywanie zapytań w stylu LINQ- **SELECT WHERE FROM:**

```
Dict<string, int> dictionary = new Dict();  
dictionary["Kamil"] = 20;  
dictionary["Marta"] = 17;
```

```
List<Pair<string, int>> rezultat = from para in dictionary  
                                where para.second > 18  
                                select new Pair({para.first, para.second});
```

i) Iterowanie po elementach słownika

```
Dict<string, int> dictionary = new Dict();  
dictionary["Kamil"] = 20;  
dictionary["Marta"] = 17;
```

```
for (Pair<string, int> para in dictionary) {  
    print(para);  
}
```

Istnieje też druga metoda, gdzie zadajemy z góry kolejność

```
function int sort_elements(Pair<string, int> para_elementow){  
    return para_elementow.second  
}
```

```
for (Pair<string, int> para in dictionary, key=sort_elements) {  
    print(para);  
}
```

Wykorzystujemy do tego funkcję, w tym przypadku jest to napisana przeze mnie funkcja `sort_elements()`, która w konsekwencji wymusza kolejność- sortuje po wartościach słownika.

6. Typ List:

- a) Lista jest
- b) Dostępne atrybuty:
 - length – zwraca ilość elementów (długość) w liście
 - type- zwraca typ listy (typ elementów w niej)
- a) Konstruktor:
 - List<typ> nazwa_zmiennej = new List();
 - List<typ> nazwa_zmiennej = new List(lista); // np. lista w postaci [1, 2]
- b) Dostęp do elementów listy za pomocą indeksu- metoda **at**:
Jeżeli nie ma elementu o danym indeksie, zostaje zgłoszony błąd.
 - nazwa_zmiennej.at(indeks)
- c) Dodanie nowego elementu do listy- metoda **append**:
Jeżeli dodawany element nie zgadza się z typem przechowywanym w liście, zostaje zgłoszony błąd.
 - nazwa_zmiennej.append(wartosc)
- d) Usunięcie elementu z listy- metoda **remove**:
Usuwa pierwsze wystąpienie danego elementu w liście, jeżeli element nie istnieje w liście zostaje zgłoszony błąd.
 - nazwa_zmiennej.remove(wartosc)

Przykłady:

```
List<string> imiona = new List();  
imiona.append("Kasia");  
imiona.append("Basia");  
imiona.append("Kamil");
```

```
print(imiona.at(0)) //wypisana zostanie imie "Kasia"
```

```
imiona.remove("Kamil")
```

7. Typ Pair:

- a) Dostępne atrybuty:
 - first – zwraca pierwszy element pary
 - second- zwraca drugi element pary
- b) Konstruktor:
 - Pair<typ_1, typ_2> nazwa_zmiennej = new Pair();
 - Pair<typ_1, typ_2> nazwa_zmiennej = new Pair([wartosc_1, wartosc_2])
- c) Przypisywanie wartości elementom pary:
 - nazwa_zmiennej.**first** = nowa_wartosc;
 - nazwa_zmiennej.**second** = nowa_wartosc;
- d) Dostęp do elementów w parze:
 - nazwa_zmiennej.**first** -> zwraca pierwszy element pary
 - nazwa_zmiennej.**second** -> zwraca drugi element pary

Przykłady:

```
Pair<string, string> para_1 = new Pair();
```

```
Pair<string, string> para_2 = new Pair(["Kamil", "Basia"]);
```

```
pair_1.first = "Antonina";
```

```
pair_1.second = "Grzegorz";
```

```
string imie_21 = pair_2.first;    //zmiennej imie_21 zostanie przypisana  
                                  //wartość "Kamil"
```

```
string imie_22 = pair_2.second;    //zmiennej imie_22 zostanie przypisana  
                                   //wartość "Basia"
```

8. Obsługa komentarzy:

- a) Cała linia począwszy od znaku "//":
 - // komentarz 1
 - // komentarz 2
 - // komentarz 3

9. Zmienne- przypisywanie do nich wartości i odczytywanie ich:

a) Semantyka obsługi zmiennych:

- Typowanie statyczne
- Typowanie silne
- Zmienne są mutowalne

b) Zakresy widoczności zmiennych

- **lokalny zakres**

```
function void funkcja_1() {  
    int a = 10;  
    print(a)  
}
```

Zmienna zdefiniowana wewnątrz ciała funkcji, jest dostępna tylko w tym bloku.

- **Zakres widoczności bloku**

```
function int main() {  
    int a = 10;  
    {  
        int b = 7;  
        print(b);  
    }  
    print(a);
```

print(b); // dostaniemy błąd, ponieważ zmienna b jest tu poza

blokiem

```
    return 0;
```

```
}
```

10. Instrukcja warunkowa if

```
int zmienna = 4;
```

```
if (zmienna == 4) {  
    print("Tak");
```

```
}
```

```
else {  
    print("Nie");
```

```
}
```

```
if (zmienna < 0) { print("Ujemna"); }  
else if (zmienna == 0) { print("Zero"); }  
else if (zmienna > 0) { print("Dodatnia"); }
```

```
if (zmienna <= 0) { print("Niedodatnia"); }  
if (zmienna >= 0) { print("Nieujemna"); }
```

11. Instrukcja pętli for

Pętla **for** w moim języku będzie podobna składnią do innych języków programowania.

```
for ( typ_zmiennej zmienna : iterowalna_zmienna) {  
    //blok instrukcji  
    //mamy tutaj dostęp do zmiennej zmienna – elementy iterowalna_zmienna  
}
```

```
Dict<string, int> imiona_lata = Dict();  
wiek[“Kasia”] = 21;  
wiek[“Basia”] = 22;
```

```
for (Pair<string, int> para_im in imiona_lata) {  
    print(“Imie: ”+ para_im.first);  
    print(“Wiek: ”+ para_im.second);  
}
```

12. Własne funkcje

a) Definiowanie własnej funkcji

```
function zwracany_typ nazwa_funkcji(lista_argumentów){  
    //blok instrukcji  
    return wartosc;  
}
```

Możliwe jest także definiowanie funkcji, które nie zwracają żadnej wartości- wtedy jako zwracany_typ używamy **void**.

Przykłady:

```
function int dodawanie(int a, int b) {  
    int c = a + b;  
    return c;  
}
```

```
function void przywitanie(string imie) {  
    print(“Witaj ” + name + “!”);  
}
```

b) Przekazywanie argumentów do funkcji przez wartość

c) Język umożliwia rekursywne wywołania funkcji

13. Gramatyka:

Gramatyka została zamieszczona w oddzielnym pliku tekstowym-
gramatyka.txt

14. Obsługa błędów:

- a) Błąd składniowy

Przykład:

```
int a = 10;  
if (a == 10) { print("10");
```

[ERROR] Syntax error: missing “}” at line 23

```
int przywitanie() {  
    print("Hello world!")  
}
```

[ERROR] Syntax error: missing “;” at line 42

- b) Błąd dzielenia przez 0

Przykład:

```
print(10 / 0);
```

[ERROR] Dividing by zero: 10/0 at line 32

- c) Błąd w indeksowaniu

```
list[int] = new list([1,2,3])  
print(list.at(4))
```

[ERROR] Index error: list index out of range at line 87

- d) Błąd w konwersji typów

```
string imie = "Kacper";  
int imie_int = int(imie);
```

[ERROR] Value error: invalid literal for int(): “kacper”

- e) Zła ilość argumentów przekazanych do funkcji:

```
function void przywitanie(string imie) {  
    print("Witaj" + imie + "!");  
}
```

```
przywitanie("Kacper", "Murygin")
```

[ERROR] Function error: Wrong number of arguments- 2 instead of 1:

line 5

15. Testowanie:

Do testowania wykorzystam testy jednostkowe, napisane przy użyciu frameworka pytest.

16.Sposób uruchomienia:

Uruchamiamy poprzez komendę:

```
python3 interpreter.py <sciezka do pliku>
```

17.Przykładowy kod

```
function int sort_elements(Pair<string, int> para_elementow){  
    return para_elementow.first;  
}
```

```
function int main(){
```

```
    int liczba_a = 0;
```

```
    Dict<string, float> kursy_walut = new Dict();
```

```
    kursy_walut["euro"] = 4.8;
```

```
    kursy_walut["dolar"] = 4.0;
```

```
    kursy_walut["jen"] = 2.0;
```

```
    //pętla for wypisująca zawartość słownika
```

```
    for (Pair<string, float> kurs: kursy_walut) {
```

```
        print("Name: " + kurs.first);
```

```
        print("Value: " + kurs.second);
```

```
    }
```

```
    //pętla for wypisująca zawartość słownika, w zadanej kolejności
```

```
    for (Pair<string, float> kurs: kursy_walut, key= {
```

```
        print("Name: " + kurs.first);
```

```
        print("Value: " + kurs.second);
```

```
    }
```

```
}
```

```
    //funkcja służąca do liczenia silni, wykorzystuje rekurencję
```

```
function int silnia(int liczba){
```

```
    if ( liczba <= 1) {
```

```
        return 1;
```

```
    }
```

```
    else {
```

```
        return liczba * silnia(liczba - 1);
```

```
    }
```

```
}
```