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 niefunkcjonalne:**
   1. Program musi zawierać funkcję main ze zwracanym typem int
   2. Każda linia kodu kończy się średnikiem- „;”
   3. Język pozwala na pisanie komentarzy
   4. Można definiować własne funkcje, wywoływać je rekurencyjnie
2. **Obsługa podstawowych typów danych liczbowych:**
   1. Rodzaje danych liczbowych:

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

1. **Obsługa typu znakowego:**
   1. Typ znakowy **string**
   2. Dozwolone operacje- konkatenacja
   3. String może zawierać dowolne znaki, w tym:

* Wyróżnik stringa- cudzysłów
* Znak nowej linii
* Znak tabulacji

1. **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.

1. **Typ słownika Dict**
   1. Dostępne atrybuty:
      * length- rozmiar słownika (ilość par klucz-wartość)
   2. Stworzenie nowej zmiennej o typie Dict- pusty słownik bez zawartości:
      * Dict<typ\_klucza, typ\_wartosci> nazwa\_zmiennej = new Dict();
   3. 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});

* 1. Dodawanie elementu do słownika:
     + nazwa\_zmiennej[klucz] = wartosc;

dict\_1[“rok”] = 2024;

* 1. Usuwanie elementu ze słownika – usuwany element o podanym kluczu:
     + nazwa\_zmiennej.delete(klucz);

dict\_1.delete(“rok”);

* 1. Wyszukiwanie elementu według klucza:
     + nazwa\_zmiennej.get(wartosc\_klucza);

dict\_data.get(“rok”);

* nazwa\_zmiennej[klucz];

dict\_data[“rok”];

* 1. 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

* 1. 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});

* 1. 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.

1. **Typ List:**
   1. Lista jest
   2. Dostępne atrybuty:

* length – zwraca ilość elementów (długość) w liście
* type- zwraca typ listy (typ elementów w niej)

1. Konstruktor:

* List<typ> nazwa\_zmiennej = new List();
* List<typ> nazwa\_zmiennej = new List(lista); // np. lista w postaci [1, 2]

1. Dostęp do elementów listy za pomocą indeksu- metoda **at**:

Jeżeli nie ma elementu o danym indeksie, zostaje zgłoszony błąd.

* nazwa\_zmiennej.at(indeks)

1. 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)

1. 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”)

1. **Typ Pair:**
   1. Dostępne atrybuty:

* first – zwraca pierwszy element pary
* second- zwraca drugi element pary
  1. Konstruktor:
* Pair<typ\_1, typ\_2> nazwa\_zmiennej = new Pair();
* Pair<typ\_1, typ\_2> nazwa\_zmiennej = new Pair([wartosc\_1, wartosc\_2])
  1. Przypisywanie wartości elementom pary:
     + nazwa\_zmiennej**.first** = nowa\_wartosc;
     + nazwa\_zmiennej**.second** = nowa\_wartosc;
  2. 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”

1. **Obsługa komentarzy:**
   1. Cała linia począwszy od znaku “//”:

// komentarz 1

// komentarz 2

// komentarz 3

1. **Zmienne- przypisywanie do nich wartości i odczytywanie ich:**
   1. Semantyka obsługi zmiennych:
   * Typowanie statyczne
   * Typowanie silne
   * Zmienne są mutowalne
   1. 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;

}

1. **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”); }

1. **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);

}

1. **Własne funkcje**
   1. 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 + “!”);

}

* 1. Przekazywanie argumentów do funkcji przez wartość
  2. Język umożliwia rekursywne wywołania funkcji

1. **Gramatyka:**

Gramatyka została zamieszczona w oddzielnym pliku tekstowym- gramatyka.ebnf

1. **Obsługa błędów:**
   1. 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

* 1. Błąd dzielenia przez 0

Przykład:

print(10 / 0);

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

* 1. 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

* 1. Błąd w konwersji typów

string imie = “Kacper”;

int imie\_int = int(imie);

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

* 1. 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

1. **Testowanie:**

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

1. **Sposób uruchomienia:**

Uruchamiamy poprzez komendę:

python3 interpreter.py <sciezka do pliku>

1. **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);

}

}