1 Wprowadzenie

EconLang to prosty DSL do budowania obliczen ekonomicznych, pracy z danymi tabelarycznymi i wykonywania zapytan SQL bezposrednio z poziomu kodu. Jezeli znasz SQL, podstawy Pythona lub VBA, szybko odnajdziesz sie w skladni.

2 Pierwszy program

Ponizszy przyklad laduje dane z CSV i liczy srednia wartosc kolumny `price`:

```
let trades = load_csv("data/trades.csv", ["ts", "price", "volume"]);
let avg_price = mean(trades.price);
Print("Srednia cena:", avg_price);
```

3 Podstawy skladni EconLang

- **Instrukcje** koncza sie srednikiem `;` (bloki `if/for/while/function` to wyjatek).
- **Bloki** otaczamy klamrami `{ ... }` i nie wymagaja srednika.
- **Komentarze**: liniowe `// komentarz` oraz blokowe `/* ... */`.
- **Identyfikatory**: litera lub `_` na poczatku, dalej litery, cyfry, `_`.
- **Typy proste**: liczby, lancuchy `"tekst"`, daty `YYYY-MM-DD`, tablice `[1,2,3]`.
- **Wstrzykiwanie do SQL**: w zapytaniach uzywaj `@zmienna`, aby przekazac wartosc z Pythona/EconLang.

3.1 Schemat programu w 5 liniach

```
// dane -> SQL -> wynik
let df = load_csv("data.csv", ["id", "price"]);
let cheap = select * from df where price < 50;
function spread(a,b) { return b - a; }
Print("Spread", spread(mean(cheap.price), 0));
```

4 Typy danych i literały

- **Liczby**: `42`, `3.14`.
- **Lancuchy**: podwojny cudzyslow, np. `"tekst"`.
- **Daty**: format `YYYY-MM-DD`, np. `2024-12-31`.
- **Tablice**: `[1, 2, 3]`, moga zawierac wyrazenia.
- **Serie czasowe**: operatory `agg_mean()`, `filter()`, `shift()` (szczegoly w rozdziale 10).
- **Wartosci z SQL**: wynik `select` to tabela (DataFrame) do dalszych obliczen.

5 Zmienne i wyrazenia

Zmienne deklaruje sie slowem `let` lub przypisaniem:

```
let limit = 100;
let note = "start";
ratio = price / 10 + 5;
```

Wyrazenia moga laczyc operatory arytmetyczne `+ - * /`, porownania `== != < <= > >=`, operator `in`, `between`, `like`/`ilike`, testy `is null` / `is not null`, oraz `exists(subquery)`.

6 Instrukcje sterujace

6.1 If / else

```
if (mean(trades.price) > 50) {
  Print("Rynek drogi");
} else {
  Print("Rynek tani");
}
```

6.2 Petle for i while

```
for i in [1,2,3] {
  Print("Iteracja", i);
}

while (balance > 0) {
  balance = balance - 10;
}
```

7 Funkcje uzytkownika

Definicja funkcji uzywa slowa `function` oraz instrukcji `return`:

```
function spread(a, b) {
  let mid = (a + b) / 2;
  return b - mid;
}

let s = spread(101, 99);
```

8 SQL

EconLang embeduje czysty SQL w wyrazeniach. Zapamietaj podstawowy szablon:

```
select [distinct] kolumny
from zrodlo
[join inna_tabela on warunek]
[where filtr]
[group by kolumny]
[having warunek_agregatu]
[order by kolumny]
[limit n offset m];
```

8.1 Przyklady minimalne

```
// Filtr z parametrem z kodu
let limit_px = 50;
let cheap = select * from df where price < @limit_px;

// Grupowanie i sortowanie
let stats = select symbol, avg(price) as avg_px, sum(volume) as vol
        from trades
        group by symbol
        having sum(volume) > 10000
        order by vol desc;

// Joiny - najczestszy wzorzec
let joined = select o.id, t.price
         from orders o
         join trades t on o.id = t.id;
```

8.2 Co znaczą poszczególne czesci

- **FROM**: tabela/alias lub wynik `load_csv` zarejestrowany jako tabela.
- **WHERE**: filtruje wiersze przed agregacja; uzywa operatorow `=, !=, <, >, between, like/ilike`.
- **GROUP BY + HAVING**: grupuje i filtruje po agregatach (np. `having sum(x) > 0`).
- **JOIN**: laczy tabele: `inner` (przeciecie), `left/right/full` (zachowanie stron), `cross` (iloczyn), `natural` (po wspolnych kolumnach).
- **ORDER BY**: `order by col asc|desc` z opcjonalnym `nulls first/last`.
- **LIMIT/OFFSET**: ogranicza wynik `limit n offset m`; przy exploracji zawsze dodaj limit.
- **Set operations**: `union [all]`, `intersect`, `except`.
- **Zmienne**: `@var` wstawia wartosc; `@func(x)` rejestruje i woła funkcje Pythona jako UDF.

8.3 Select items i aliasy

```
// Gwiazdka lub lista wyrazen
select *, price * 1.23 as gross from trades;

// Dostep do kolumn przez kropke lub alias tabeli
select t.id, t.price from trades t;
```

8.4 Select items i aliasy


```
// Gwiazdka lub lista wyrazen
select *, price * 1.23 as gross from trades;

// Dostep do kolumn przez kropke lub alias tabeli
select t.id, t.price from trades t;
```


9 Operacje agregujace i grupowanie

```
let stats = select symbol, avg(price) as avg_px, sum(volume) as vol
          from trades
          group by symbol
          having sum(volume) > 10000
          order by vol desc;
```

extbfDostepne agregaty: `avg, min, max, sum, count, mean`. W SQL mozesz takze uzyc funkcji uzytkownika zarejestrowanych w Pythonie (np. `@myfunc(x)`).

10 Tablice i serie

Tablice w kodzie moga sluzyc do iteracji, filtrowania lub do budowy okien czasowych.

```
let nums = [1,2,3];
let shifted = price.shift(1);   // przesuniecie o 1
let cleaned = price.filter(100); // obciecie do pierwszych 100 elementow
let rolling = price.agg_mean(from 2024-01-01 to 2024-12-31);
```

Skrotowe operatory na seriach:

- `id agg_mean(window)` lub `agg_sum`, `agg_std`.
- `id.filter(n)` – zwraca pierwsze `n` elementow.
- `id.shift(n)` – przesuwa serie o `n` krokow.
- Okno opcjonalnie opisane `from <data> to <data>`.

11 Case, between, in, like

```
let flag = case
  when price > 100 then "wysoki"
  when price between 50 and 100 then "sredni"
  else "niski"
end;

let filtered = select * from trades
  where symbol in ("AAPL", "MSFT")
    and note ilike "%earnings%";
```

12 Uzycie funkcji runtime

Najczesciej uzywane funkcje z `econ_runtime.py`:

- **Statystyka**: `mean`, `avg`, `min`, `max`, `sum`, `count`, `variance`, `std`.
- **Finanse**: `irr`, `npv`, `cagr`, `sharpe_ratio`, `value_at_risk`.
- **Czas**: `date()`, `now()`, `dateadd()`, `datediff()`.
- **Tekst**: `len`, `replace`, `lowercase`, `uppercase`, `trim`.
- **Magiczne wyjscia**: `Print`, `DisplayTable`, `DisplayChart` kolekcjonuja dane do UI.

13 Forecast i indykatory

```
let forecasted = forecast_arima(prices, p=2, d=1, q=2);
let rsi_val = indicator_rsi(prices, period=14);
```

Funkcje te sa wykonywane po stronie Pythona, wiec moga byc uzyte zarowno w kodzie, jak i w zapytaniach SQL poprzez `@func(args)`.

14 Najczestsze wzorce

- **Filtrowanie tabeli**: `select * from t where cond;`.
- **Join dwoch zrodel**: `from a join b on a.id = b.id`.
- **Agregacja po kolumnie**: `group by col` i agregaty.
- **Uzycie zmiennej w SQL**: `where price > @limit`.
- **Transformacja kolumny**: `select price * 1.23 as gross`.

15 Skrocona sciaga skladni

extbfDeklaracja	`let x = 1;`
extbfPrzypisanie	`x = x + 1;`
extbfFunkcja	`function f(a,b) { return a+b; }`
extbfIf / else	`if (cond) { ... } else { ... }`
extbfFor	`for i in [1,2,3] { ... }`
extbfWhile	`while (cond) { ... }`
extbfCase	`case when a>0 then 1 else 0 end`
extbfSQL select	`select col from t where col > 0;`
extbfJoin	`from a join b on a.id = b.id`
extbfGroup	`group by col having sum(x)>0`
extbfLimit/offset	`limit 10 offset 5`
extbfZmienne w SQL	`where price > @limit`

16 Rozszerzenia i UDF

Kazda funkcja Pythona dostepna w zrodle uzytkownika moze zostac wywolana w SQL przez zapis `@nazwa(args)`. Transpiler automatycznie rejestruje takie funkcje jako UDF w DuckDB.