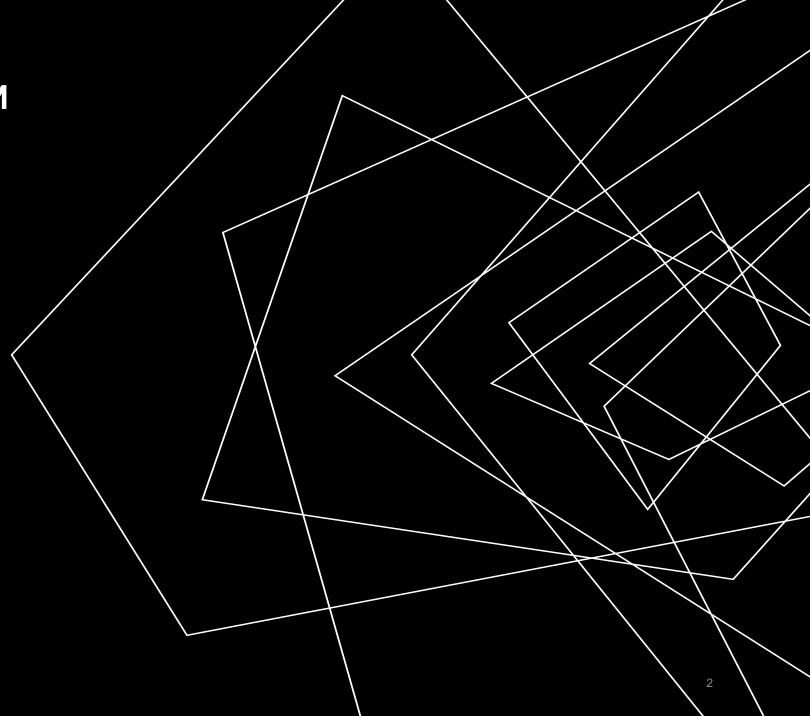


## ПЛАН ПРЕЗЕНТАЦИИ

- Условные конструкции
- Множественный выбор
- Циклы
- Обработка ошибок
- Отладка программы
- Проверки и утверждения
- Список литературы



### УСЛОВНЫЕ КОНСТРУКЦИИ

- if (if [условие] [then-выражение] [else-выражение])
   Пример: (if (> x 0) "positive" "negative")
- when (when [условие] [тело])
   Пример: (when debug (print "Debug mode is active"))
- unless противоположность when

```
(defun my-file-exists-p (file)
(probe-file file)); Функция-заглушка для проверки существования файла
(let ((file "nonexistent.txt"))
(unless (my-file-exists-p file)
(print (format nil "File ~a does not exist, creating..." file))))

Вывод: "File nonexistent.txt does not exist, creating..."
```

#### **МНОЖЕСТВЕННЫЙ ВЫБОР**

• cond – последовательная проверка условий

```
29 (defun number-type (n)
30 (cond ((> n 0) "positive")
31 ((< n 0) "negative")
32 (t "zero")))
33 (print (number-type 10)) ; Вывод: "positive"
34 (print (number-type -5)) ; Вывод: "negative"
35 (print (number-type 0)) ; Вывод: "zero"
```

• case – выбор по конкретным значениям

```
(defun day-of-week (day)

(case day

(1 "Monday")

(2 "Tuesday")

(3 "Wednesday")

(otherwise "Unknown day")))

(print (day-of-week 1)); Вывод: "Monday"

(print (day-of-week 4)); Вывод: "Unknown day"
```

## ЦИКЛЫ

• **loop** – мощный макрос для сложных циклов с ключевыми словами (for, while и т.д.)

```
51 (defun sum-squares (n)
52 (loop for i from 1 to n
53 | sum (* i i)))
54 (print (sum-squares 3)); Вывод: 14
```

dotimes — цикл по числам: (dotimes (i count [результат]) [тело])

```
57 (dotimes (i 5)
58 (print i)) ; Вывод: 0 1 2 3 4
```

• dolist – цикл по элементам списка: (dolist ([элемент] [список] [результат]) [тело])

```
61 (dolist (item '(a b c))
62 (print item)) ; Вывод: a b c
```

• do — универсальный цикл: (do (список-инициализаций) (условие-выхода) [тело])

```
65 (defun factorial-do (n)

66 (do ((i 1 (1+ i))

67 (result 1 (* result i)))

68 ((> i n) result)))

69 (print (factorial-do 5)) ; Вывод: 120 (5!)
```

#### ОБРАБОТКА ОШИБОК

```
(defun divide-safe (a b)

(if (zerop b)

(error "Division by zero")

(/ a b)))

(handler-case

(print (divide-safe 10 0)); Вызовет ошибку

(error (e) (format t "Caught error: ~a~%" e)))

Вывод: "Caught error: Division by zero"

(ргіпт (ignore-errors: Игнорирование ошибок

(ргіпт (ignore-errors (divide-safe 10 0))); Вывод: NIL
```

- error выбрасывает ошибку
- ignore-errors игнорирует ошибки и возвращает nil
- handler-case ловит и обрабатывает ошибку

• unwind-protect – обеспечивает выполнение cleanup-кода даже при ошибке

```
(defun cleanup-example ()
         (unwind-protect
              (progn
                (print "Performing operation...")
               (error "An error occurred"))
100
          (print "Cleaning up...")))
101
102
       (handler-case
103
          (cleanup-example)
         (error (e) (format t "Caught error: ~a~%" e))) ;
104
          Вывод: "Performing operation..." "Cleaning up..." "Caught error: An error occurred"
105
```

### ОТЛАДКА ПРОГРАММЫ

• trace и untrace – трассировка функций для просмотра вызовов и аргументов

```
111 (defun my-square (x)
112 (* x x))
113 (trace my-square)
114 (print (my-square 4)) ; Вывод: трассировка вызовов my-square и результат 16
115 (untrace my-square) ; Отключение трассировки

0: (MY-SQUARE 4)
0: MY-SQUARE returned 16
16
```

• break – вставляет точку останова

```
(defun break-example (x)
(break "Breakpoint at x=~a" x)
(* x 2))
(break-example 5)

Breakpoint at x=5
debugger invoked on a SIMPLE-CONDITION...
```

0: [CONTINUE] Continue execution

restarts:

#### ОТЛАДКА ПРОГРАММЫ

• **step** – пошаговая отладка

```
Evaluating call to MY-SQUARE in frame 0 with arguments (4):
Step into call to MY-SQUARE? [y/n] y
Evaluating (* X X) in frame 1:
Step into (* X X)? [y/n] y
Evaluating X in frame 2:
Step into X? [y/n] n
Evaluating X in frame 2:
Step into X? [y/n] n
Result of (* X X) is 16
Result of MY-SQUARE is 16
16
```

#### проверки и утверждения

• assert – проверка условий во время выполнения

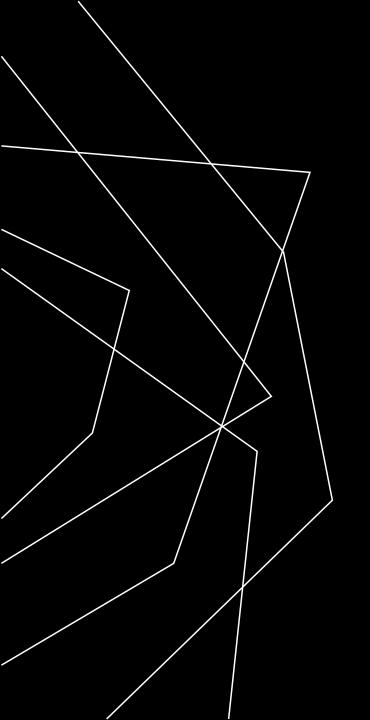
```
122
      (defun safe-divide (a b)
        (assert (not (zerop b)) (b) "Cannot divide by zero")
123
        (/ a b))
124
      (print (safe-divide 10 2)) ; Вывод: 5
125
      (handler-case
126
127
          (print (safe-divide 10 0)) ; Вызовет ошибку
128
        (error (e) (format t "Caught error: ~a~%" e)))
        Вывод: "Caught error: Cannot divide by zero"
129
```

• **check-type** – проверка типа переменной

```
131
      (defun process-number (x)
        (check-type x number "a number")
132
        (* x 2))
133
134
      (print (process-number 5)) ; Вывод: 10
135
      (handler-case
136
          (print (process-number "string")) ; Вызовет ошибку
137
        (error (e) (format t "Caught error: ~a~%" e)))
138
      ; Вывод: "Caught error: The value string is not of type number"
```

### СПИСОК ЛИТЕРАТУРЫ

- LISP Краткое руководство
- Основы программирования на языке Lisp



# СПАСИБО ЗА ВНИМАНИЕ