Теория Формальных Языков.

Лабораторная работа №4.

Вариант — 11.

Масягин М. М.

ИУ9-52Б

Задание-3.

Провести сравнительный анализ синтаксиса конструкций If , While и способа определения переменных в Python от С#.

1) Способ определения переменных:

# ● Python — это объектно-ориентированный язык со строгой динамической неявной типизацией. Явно указывать тип значения переменной нет необходимости, поэтому объявление новой переменной происходит следующим образом:

*>>> a = 5*

*>>> a*

*5*

*>>> b = "kek"*

*>>> b*

*"kek"*

*>>> c = float(5) # без явного приведения типа c будет int*

*>>> c*

*5.0*

*>>> d = list(1, 2, 3, 4, 5.0)*

*>>> d*

*[1, 2, 3, 4, 5.0] # первые 4 элемента — int, 5 — float*

Начиная с ранних версий Python 3 существует возможность указывать типы, а с Python 3.6 и объявляемых переменных, что тем не менее ничего не гарантирует.

*>>> def sqr(a: float) → float:*

*>>> ...*

*>>> e: int = "lol"*

*>>> e*

*"lol"*

● C# - это объектно-ориентированный язык со строгой статической обычно явной (почему обычно — будет показано далее) типизацией.

*int f = 42;*

*var g = 5.0; (компилятор автоматически выведет тип переменной)*

Тем не менее, так как помимо C# платформа .NET поддерживает и множество других языков (например, F#), то в C# все же есть возможность создать переменную с динамической типизацией, что делать крайне не рекомендуется.

*Dynamic h = 7;*

*h.hello(); // компилятор не выдаст никакой ошибки, но в рантайме программа упадет*

2) If:

● Python не использует «;» и «{}» - классические разделительные операторы в C-подобных языках, к коим относится и C#. Выделение блоков кода строится на основе пробелов или табуляций (первое предпочтительнее). Также в условиях не требуются круглые скобки.

Оператор if в Python помимо классических if и else имеет также и ветвь elif (сокращение от else if):

if a == 5:

print(*"a = 5")*

*elif a == 6:*

*print("a = 6")*

*else if a == 7:*

*print("a = 5")*

*else:*

*print("azazaza")*

*print("a = 8")*

● C# - это классический С-подобный язык и оператор If выглядит в нем соответствующе:в

*if (a == 5)*

*console.WriteLine("a = 5")*

*else if (a == 6)*

*console.WriteLine("a = 6")*

*else*

*console.WriteLine("a = 7")*

Для того, чтобы разместить в одном блоке несколько операций, требуется использовать фигурные скобки.

*if (a == 5) {*

*cconsole.WriteLine("a = 5")*

*console.WriteLine("a equals to five")*

*}*

3) While:

● Оператор While (также как и For) в языке Python интересен тем, что он имеет else ветвь, которая выполняется, если в цикле не был вызван break. В остальном Pythonовский While вполне обычен. Помимо break есть и continue.

*>>> i = 0*

*>>> while i < 5:*

*>>> i += 1*

*>>> if i == 5:*

*>>> break*

*>>> else:*

*>>> print("5 not found")*

Важно отметить, что цикла Do-While в Python нет и его приходится «имитировать» посредством while True: с проверкой с breakом в конце.

● Оператор While в языке C# типичен для С-подобных языков и не представляет из себя ничего интересного. Помимо него имеется и такой же классические Do-While.

*int i = 0;*

*while (i < 5) {*

*Console.WriteLine(i);*

*i++;*

*}*

*int i = -1;*

*do {*

*Console.WriteLine(i);*

*i--;*

*}*

*while (i > 0);*

*Задание-1.*

*Построение грамматики.*

***# Общая структура программы***

*program -> source\_elements*

*source\_elements -> empty | source\_element\_list*

*source\_element\_list -> source\_element | source\_element\_list*

*source\_element*

*source\_element -> statement | function\_declaration*

*statement -> block | variable\_statement | empty\_statement*

*| expr\_statement | if\_statement | iteration\_statement | continue\_statement | break\_statement | return\_statement | with\_statement | switch\_statement | labelled\_statement | throw\_statement | try\_statement | debugger\_statement | function\_declaration*

*block -> LBRACE source\_elements RBRACE*

*literal -> null\_literal | boolean\_literal | numeric\_literal |*

*string\_literal | regex\_literal*

*boolean\_literal -> TRUE | FALSE*

*null\_literal -> NULL*

*numeric\_literal -> NUMBER*

*string\_literal -> STRING*

*regex\_literal -> REGEX*

*identifier -> ID*

***# Выражения***

*primary\_expr -> primary\_expr\_no\_brace | object\_literal*

*primary\_expr\_no\_brace -> identifier | literal | array\_literal |*

*LPAREN expr RPAREN | THIS*

*array\_literal -> LBRACKET elision\_opt RBRACKET | LBRACKET*

*element\_list RBRACKET | LBRACKET element\_list COMMA elision\_opt RBRACKET*

*element\_list -> elision\_opt assignment\_expr | element\_list COMMA*

*elision\_opt assignment\_expr*

*elision\_opt -> empty*

*elision\_opt -> elision*

*elision -> COMMA | elision COMMA*

*object\_literal -> LBRACE RBRACE | LBRACE property\_list RBRACE |*

*LBRACE property\_list COMMA RBRACE*

*property\_list -> property\_assignment | property\_list COMMA*

*property\_assignment*

***# GET / SET***

*property\_assignment -> property\_name COLON assignment\_expr |*

*GETPROP property\_name LPAREN RPAREN LBRACE function\_body RBRACE | SETPROP property\_name LPAREN formal\_parameter\_list RPAREN LBRACE function\_body RBRACE*

*property\_name -> identifier | string\_literal | numeric\_literal*

***# Left-Hand-Side Expressions***

*member\_expr -> primary\_expr | function\_expr | member\_expr LBRACKET*

*expr RBRACKET | member\_expr PERIOD identifier | NEW member\_expr arguments*

*member\_expr\_nobf -> primary\_expr\_no\_brace | function\_expr |*

*member\_expr\_nobf LBRACKET expr RBRACKET | member\_expr\_nobf PERIOD identifier | NEW member\_expr arguments*

*new\_expr -> member\_expr | NEW new\_expr*

*new\_expr\_nobf -> member\_expr\_nobf | NEW new\_expr*

*call\_expr -> member\_expr arguments | call\_expr arguments |*

*call\_expr LBRACKET expr RBRACKET | call\_expr PERIOD identifier*

*call\_expr\_nobf -> member\_expr\_nobf arguments | call\_expr\_nobf*

*arguments | call\_expr\_nobf LBRACKET expr RBRACKET | call\_expr\_nobf PERIOD identifier*

*arguments -> LPAREN RPAREN | LPAREN argument\_list RPAREN*

*argument\_list -> assignment\_expr |*

*argument\_list COMMA assignment\_expr*

*left\_hand\_side\_expr -> new\_expr | call\_expr*

*left\_hand\_side\_expr\_nobf -> new\_expr\_nobf | call\_expr\_nobf*

***# Postfix Expressions***

*postfix\_expr -> left\_hand\_side\_expr | left\_hand\_side\_expr PLUSPLUS*

*| left\_hand\_side\_expr MINUSMINUS*

*postfix\_expr\_nobf -> left\_hand\_side\_expr\_nobf |*

*left\_hand\_side\_expr\_nobf PLUSPLUS | left\_hand\_side\_expr\_nobf MINUSMINUS*

***# Unary Operators***

*unary\_expr -> postfix\_expr | unary\_expr\_common*

*unary\_expr\_nobf -> postfix\_expr\_nobf | unary\_expr\_common*

*unary\_expr\_common -> DELETE unary\_expr | VOID unary\_expr | TYPEOF*

*unary\_expr | PLUSPLUS unary\_expr | MINUSMINUS unary\_expr | PLUS unary\_expr | MINUS unary\_expr | BNOT unary\_expr | NOT unary\_expr*

***# Multiplicative Operators***

*multiplicative\_expr -> unary\_expr | multiplicative\_expr MULT*

*unary\_expr | multiplicative\_expr DIV unary\_expr | multiplicative\_expr MOD unary\_expr*

*multiplicative\_expr\_nobf -> unary\_expr\_nobf |*

*multiplicative\_expr\_nobf MULT unary\_expr | multiplicative\_expr\_nobf DIV unary\_expr | multiplicative\_expr\_nobf MOD unary\_expr*

***# Additive Operators***

*additive\_expr -> multiplicative\_expr | additive\_expr PLUS*

*multiplicative\_expr | additive\_expr MINUS multiplicative\_expr*

*additive\_expr\_nobf -> multiplicative\_expr\_nobf |*

*additive\_expr\_nobf PLUS multiplicative\_expr | additive\_expr\_nobf MINUS multiplicative\_expr*

***# Bitwise Shift Operators***

*shift\_expr -> additive\_expr | shift\_expr LSHIFT additive\_expr | shift\_expr RSHIFT additive\_expr | shift\_expr URSHIFT additive\_expr*

*shift\_expr\_nobf -> additive\_expr\_nobf | shift\_expr\_nobf LSHIFT additive\_expr | shift\_expr\_nobf RSHIFT additive\_expr | shift\_expr\_nobf URSHIFT additive\_expr*

***# Relational Operators***

*relational\_expr -> shift\_expr | relational\_expr LT shift\_expr |*

*relational\_expr GT shift\_expr | relational\_expr LE shift\_expr | relational\_expr GE shift\_expr | relational\_expr INSTANCEOF shift\_expr | relational\_expr IN shift\_expr*

*relational\_expr\_noin -> shift\_expr | relational\_expr\_noin LT*

*shift\_expr | relational\_expr\_noin GT shift\_expr | relational\_expr\_noin LE shift\_expr | relational\_expr\_noin GE shift\_expr | relational\_expr\_noin INSTANCEOF shift\_expr*

*relational\_expr\_nobf -> shift\_expr\_nobf | relational\_expr\_nobf LT*

*shift\_expr | relational\_expr\_nobf GT shift\_expr | relational\_expr\_nobf LE shift\_expr | relational\_expr\_nobf GE shift\_expr | relational\_expr\_nobf INSTANCEOF shift\_expr | relational\_expr\_nobf IN shift\_expr*

***# Equality Operators***

*equality\_expr -> relational\_expr | equality\_expr EQEQ*

*relational\_expr | equality\_expr NE relational\_expr | equality\_expr STREQ relational\_expr | equality\_expr STRNEQ relational\_expr*

*equality\_expr\_noin -> relational\_expr\_noin | equality\_expr\_noin*

*EQEQ relational\_expr | equality\_expr\_noin NE relational\_expr | equality\_expr\_noin STREQ relational\_expr | equality\_expr\_noin STRNEQ relational\_expr*

*equality\_expr\_nobf -> relational\_expr\_nobf | equality\_expr\_nobf*

*EQEQ relational\_expr | equality\_expr\_nobf NE relational\_expr | equality\_expr\_nobf STREQ relational\_expr | equality\_expr\_nobf STRNEQ relational\_expr*

***# Binary Bitwise Operators***

***# Binary Logical Operators***

*logical\_and\_expr -> bitwise\_or\_expr |*

*logical\_and\_expr AND bitwise\_or\_expr*

*logical\_and\_expr\_noin -> bitwise\_or\_expr\_noin |*

*logical\_and\_expr\_noin AND bitwise\_or\_expr\_noin*

*logical\_and\_expr\_nobf -> bitwise\_or\_expr\_nobf |*

*logical\_and\_expr\_nobf AND bitwise\_or\_expr\_nobf*

*logical\_or\_expr -> logical\_and\_expr | logical\_or\_expr OR*

*logical\_and\_expr*

*logical\_or\_expr\_noin -> logical\_and\_expr\_noin |*

*logical\_or\_expr\_noin OR logical\_and\_expr\_noin*

*logical\_or\_expr\_nobf -> logical\_and\_expr\_nobf |*

*logical\_or\_expr\_nobf OR logical\_and\_expr\_nobf*

***# Conditional Operator ( ? : )***

*conditional\_expr -> logical\_or\_expr | logical\_or\_expr CONDOP*

*assignment\_expr COLON assignment\_expr*

*conditional\_expr\_noin -> logical\_or\_expr\_noin |*

*logical\_or\_expr\_noin CONDOP assignment\_expr\_noin COLON assignment\_expr\_noin*

*conditional\_expr\_nobf -> logical\_or\_expr\_nobf |*

*logical\_or\_expr\_nobf CONDOP assignment\_expr COLON assignment\_expr*

***# Assignment Operators***

*assignment\_expr -> conditional\_expr |*

*left\_hand\_side\_expr assignment\_operator assignment\_expr*

*assignment\_expr\_noin -> conditional\_expr\_noin |*

*left\_hand\_side\_expr assignment\_operator assignment\_expr\_noin*

*assignment\_expr\_nobf -> conditional\_expr\_nobf |*

*left\_hand\_side\_expr\_nobf assignment\_operator assignment\_expr*

*assignment\_operator -> EQ | MULTEQUAL | DIVEQUAL | MODEQUAL |*

*PLUSEQUAL | MINUSEQUAL | LSHIFTEQUAL | RSHIFTEQUAL | URSHIFTEQUAL | ANDEQUAL | XOREQUAL | OREQUAL*

***# Comma Operator***

*expr -> assignment\_expr | expr COMMA assignment\_expr*

*expr\_noin -> assignment\_expr\_noin | expr\_noin COMMA*

*assignment\_expr\_noin*

*expr\_nobf -> assignment\_expr\_nobf | expr\_nobf COMMA*

*assignment\_expr*

***# Variable Statement***

*variable\_statement -> VAR variable\_declaration\_list SEMI |*

*VAR variable\_declaration\_list auto\_semi*

*variable\_declaration\_list -> variable\_declaration |*

*variable\_declaration\_list COMMA variable\_declaration*

*variable\_declaration\_list\_noin → variable\_declaration\_noin |*

*variable\_declaration\_list\_noin COMMA variable\_declaration\_noin*

*variable\_declaration -> identifier | identifier initializer*

*variable\_declaration\_noin -> identifier | identifier*

*initializer\_noin*

*initializer -> EQ assignment\_expr*

*initializer\_noin -> EQ assignment\_expr\_noin*

***# Empty statement***

*empty\_statement -> SEMI*

***# Expression Statement***

*expr\_statement -> expr\_nobf SEMI | expr\_nobf auto\_semi*

*# IF Statement*

*if\_statement -> IF LPAREN expr RPAREN statement*

*if\_statement -> IF LPAREN expr RPAREN statement ELSE statement*

***# DO-WHILE Statement***

*iteration\_statement → DO statement WHILE LPAREN expr RPAREN SEMI | DO statement WHILE LPAREN expr RPAREN auto\_semi*

***# WHILE Statemet***

*iteration\_statement -> WHILE LPAREN expr RPAREN statement*

***# FOR Statement***

*iteration\_statement -> FOR LPAREN expr\_noin\_opt SEMI expr\_opt SEMI expr\_opt RPAREN statement | FOR LPAREN VAR variable\_declaration\_list\_noin SEMI expr\_opt SEMI expr\_opt RPAREN statement | FOR LPAREN left\_hand\_side\_expr IN expr RPAREN statement | FOR LPAREN VAR identifier IN expr RPAREN statement | FOR LPAREN VAR identifier initializer\_noin IN expr RPAREN statement*

*expr\_opt -> empty | expr*

*expr\_noin\_opt -> empty | expr\_noin*

***# Continue statement***

*continue\_statement -> CONTINUE SEMI | CONTINUE auto\_semi*

*continue\_statement -> CONTINUE identifier SEMI | CONTINUE identifier auto\_semi*

***# Break statement***

*break\_statement -> BREAK SEMI | BREAK auto\_semi*

*break\_statement -> BREAK identifier SEMI | BREAK identifier auto\_semi*

***# Return statement***

*return\_statement -> RETURN SEMI | RETURN auto\_semi*

*return\_statement -> RETURN expr SEMI | RETURN expr auto\_semi*

***# With statement***

*with\_statement -> WITH LPAREN expr RPAREN statement*

***# Switch Statement***

*switch\_statement -> SWITCH LPAREN expr RPAREN case\_block*

*case\_block -> LBRACE case\_clauses\_opt RBRACE | LBRACE case\_clauses\_opt default\_clause case\_clauses\_opt RBRACE*

*case\_clauses\_opt -> empty | case\_clauses*

*case\_clauses -> case\_clause | case\_clauses case\_clause*

*case\_clause -> CASE expr COLON source\_elements*

*default\_clause -> DEFAULT COLON source\_elements*

***# Labelled statement***

*labelled\_statement -> identifier COLON statement*

***# Throw Statement***

*throw\_statement -> THROW expr SEMI | THROW expr auto\_semi*

***# Try Statement***

*try\_statement -> TRY block catch |*

*TRY block finally | TRY block catch finally*

*catch -> CATCH LPAREN identifier RPAREN block*

*finally -> FINALLY block*

***# Function definition***

*function\_declaration -> FUNCTION identifier LPAREN RPAREN LBRACE*

*function\_body RBRACE | FUNCTION identifier LPAREN formal\_parameter\_list RPAREN LBRACE function\_body RBRACE*

*function\_expr → FUNCTION LPAREN RPAREN LBRACE function\_body RBRACE*

*| FUNCTION LPAREN formal\_parameter\_list RPAREN LBRACE function\_body RBRACE | FUNCTION identifier LPAREN RPAREN LBRACE function\_body RBRACE | FUNCTION identifier LPAREN formal\_parameter\_list RPAREN LBRACE function\_body RBRACE*

*formal\_parameter\_list -> identifier |*

*formal\_parameter\_list COMMA identifier*

*function\_body -> source\_elements*

*Грамматика в форме Грейбах.*

*Задание-2.*

*Пример правильной грамматики.*

*Пример неправильной грамматики.*

*Задание-4.*