

FSE598 前沿计算技术

模块 2 数据与数据处理 单元 1 编程语言基础 第 2 讲 编程语言设计

讲座的英文版内容基于本书：

Y. Chen 《编程语言入门：C、C++、Scheme、Prolog、C# 和 Python 编程》(Introduction to Programming Languages: Programming in C, C++, Scheme, Prolog, C#, and Python), 第 6 版, Kendall Hunt Publishing Company, 2019 年。
<https://www.public.asu.edu/~ychen10/book/IntroPl.html>

学习

- ❑ 程序语言结构
- ❑ 词汇和句法结构
- ❑ 上下文结构
- ❑ 类型和类型转换

程序结构

我们如何表达程序，它们的意义是什么？

不同层次的分析：

- 词法：
begin ... end, { ... }, ;
- 句法：
if ... then ... else ..., switch, for ... while ...
- 上下文：
变量、类型、名称、初始化等。
- 语义：
程序的意义和执行行为

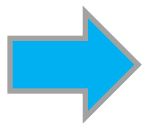
程序结构：词汇

词法符号、单位、标记：基本构建块。

- **标识符**：感兴趣的对象（变量、规程名称等）的名称（由程序员选择）。

问题：长度、区分大小写、允许的字符、特殊规则。

- **关键词**：语言设计者保留的名称：if、else、switch、int、float、char
- **运算符**：+、*、<<、>=、!、&&、||
- **分隔符**：, ; . () 等
- **字面量**：数字、字符、字符串
- **注释**：/* ... */, // ...
- **布局、空间**：有些语言是自由格式的，而有些则不是，例如 Python。



Python 代码布局结构

- ❑ 不需要大括号。
- ❑ Python 使用代码缩进（制表符或空格以及对齐来识别块）。
- ❑ 在 while 循环中，有一个 if-then-else 语句，它的 if 部分有两个语句的代码块。同样，不需要大括号，使用代码缩进定义块。

```
Python 3.7 (32-bit)

>>> # Program to generate a random number between 0 and 99 and print some
... # import the random module
... import random
>>> i= 0
>>> count = 0
>>> limit = 5
>>> while i < limit:
...     r = random.randint(0,99)
...     if r < 50:
...         ➡ count +=1
...         print("A small number is generated", r, "count is", count)
...     else:
...         print("A large number is generated", r)
... ➡ i +=1
...
A small number is generated 4 count is 1
A large number is generated 63
A large number is generated 97
A small number is generated 32 count is 2
A large number is generated 99
>>>
```

程序结构：句法

- 语言**句法**描述了如何将词汇单元组合在一起以形成有效的句子/陈述。定义标准：

- 巴科斯范式表示法
- 语法图

- 20 世纪 50 年代后期，引入了一种简洁而正式的语法表达方式：巴科斯范式表示法(BNF)。

BNF：用于描述语言的元语言，一种与上下文无关的语言。两种类型的符号：

- **非终结符号**：语法中的符号——它们不会出现在最终程序中；
- **终结符号**：出现在实际程序中的符号——无法再从语法角度翻译。

程序结构：句法

示例：定义一种非常简单语言(VSL)

```
<sentence> ::= <subject> <verb> <object>
<subject>  ::= <adjective> <noun>      | <noun>
<adjective> ::= <adjective> | <adjective> <adjective>
<object>   ::= <subject>      | <adjective>
<noun>      ::= table | horse | computer
<adjective> ::= big | fast | good | high
<verb>      ::= is | makes
<sentence>  ::= <verb> <subject> <object> ?
```

big fast computer is good table
fast big high horse is good
high is computer
horse eats grass

table makes horse
fast table is high
is fast computer good
is fast computer good?

程序结构：句法

示例：定义一种非常简单编程语言(VSPL)

<code><char></code>	<code>::=</code>	<code>a b c ... z 0 1 ... 9</code>
<code><operator></code>	<code>::=</code>	<code>+ - * / % < > == </code>
<code><variable></code>	<code>::=</code>	<code><char> <char> < variable></code>
<code>< expr ></code>	<code>::=</code>	<code>< variable> <operator><variable></code> <code> (<expr>) <operator> (<expr>)</code>
<code><assign></code>	<code>::=</code>	<code>< variable> = <expr>;</code>
<code><statements></code>	<code>::=</code>	<code><assign> <assign> <statements></code>

哪种有效？

`2sum = 2+3;`

`sum4 = (a1 + a2) * (3b % 4*b);`


`3p4rd2 = ((1a + a2) * (b3 % b4)) / (c7 - c8);`

`foo_bar = (a1 + a2 - b3 - b4);`

`(a1 / a2) = (c3 - c4);`

程序结构：句法

语言也可以由语法图定义

→ **if** (condition) statements ;  **else** statements ;

if (a < 0) **return** -a;


if (a < 0) **return** -a; **else return** a;


→ **switch** (expr) { → **case** value: statements ;  **default:** statements; } →

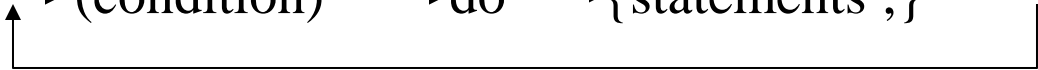
```
switch (ch) {  
    case '+': x = a + b; break;  
    case '-': x = a - b; break;  
    case '*': x = a * b; break;  
    case '÷': x = a / b; x = round(x); break;  
    default: printf("invalid operator"); break;  
}
```


"while (condition) do {statements;}"的语法图是怎样的？

(A) \rightarrow **while** \rightarrow (condition) \rightarrow do \rightarrow {statements;} \rightarrow

(B) \rightarrow **while** \rightarrow (condition) \rightarrow do \rightarrow {statements;} \rightarrow


(C) \rightarrow **while** \rightarrow (condition) \rightarrow do \rightarrow {statements ;} \rightarrow


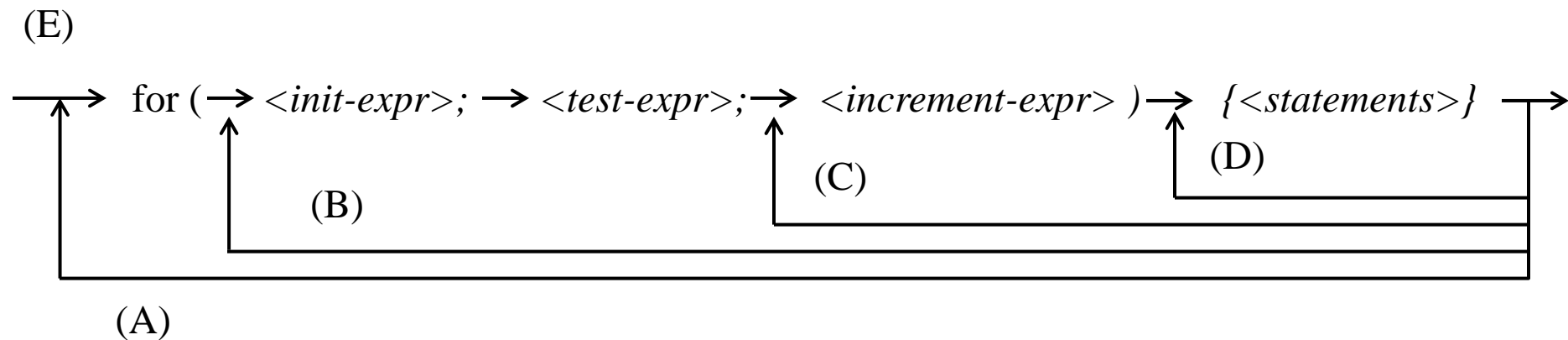
(D) \rightarrow **while** \rightarrow (condition) \rightarrow do \rightarrow {statements ;} \rightarrow


(E) \rightarrow **while** \rightarrow (condition) \rightarrow do \rightarrow {statements ;} \rightarrow


程序结构：句法

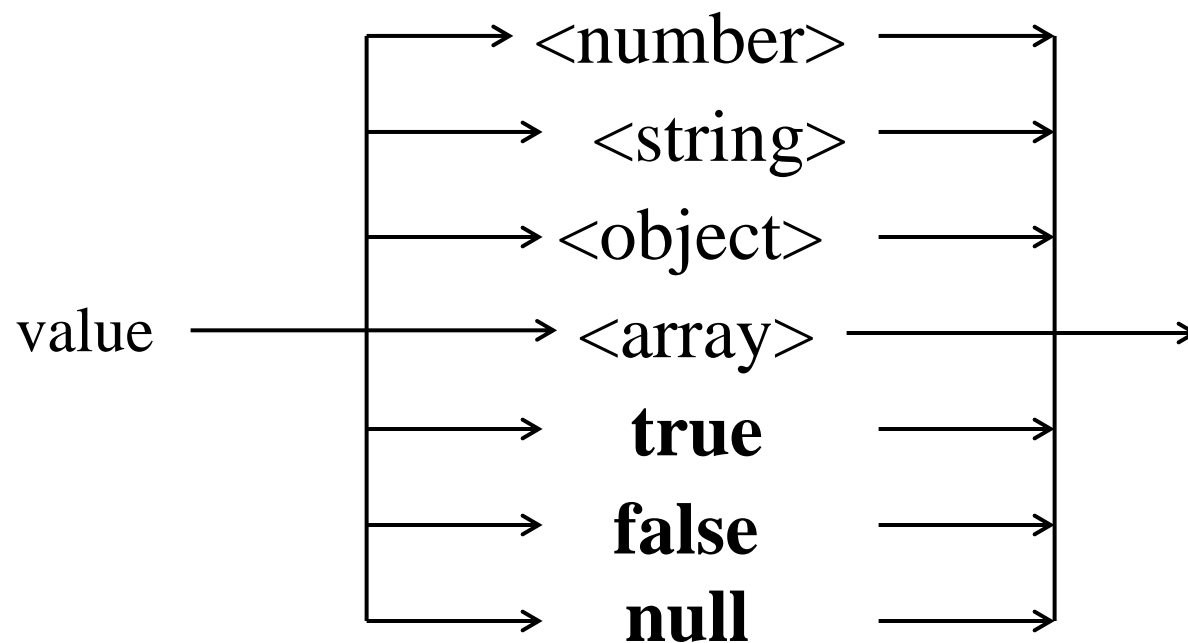
“for 循环”的语法图是怎样的？

```
for ( <init-expr>; <test-expr>; <increment-expr> )  
{  
    <statements>  
}
```



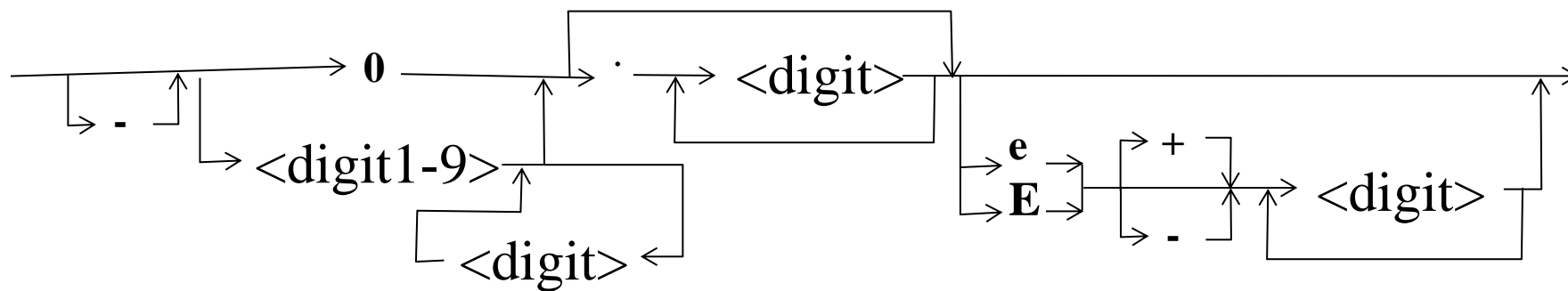
语法图案例研究：JSON（JavaScript Object Notation）

使用语法图定义数据

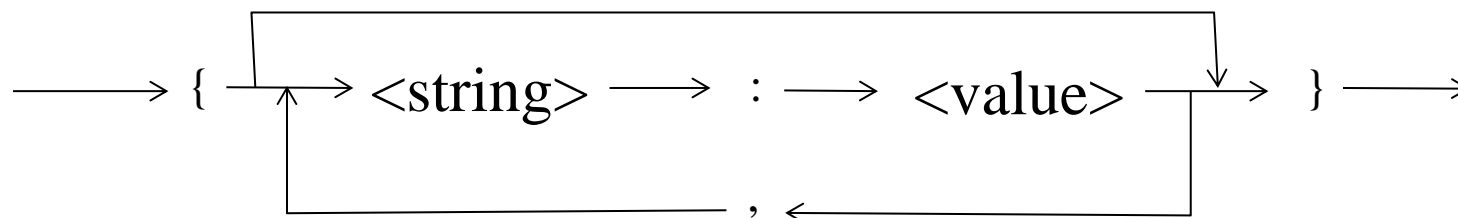


JSON 数字和对象格式

数字



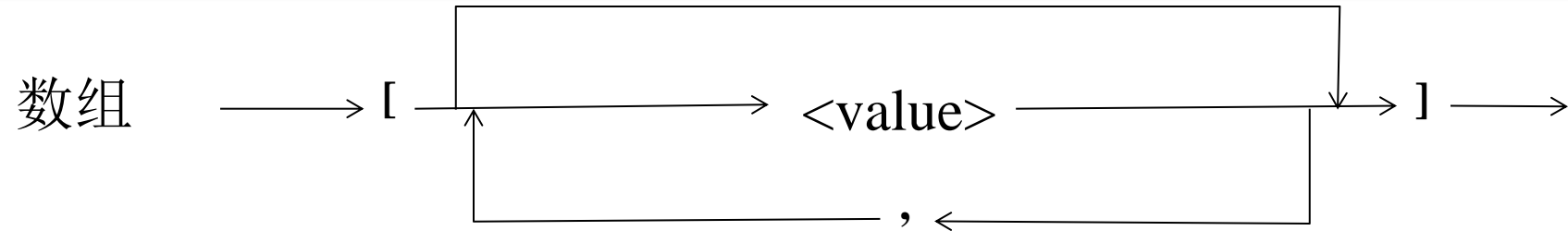
对象



对象示例

```
{"name":"John Doe", "age":25, "married":true, "University":"ASU",  
"Graduated":false,  
"Courses":{"CSE240":200,"CSE310":300,"CSE446":400,  
"GPA":3.75}}
```

JSON 数组



字符串数组

`["John Doe", "Mary", "Smith"]`

混合值数组

`[{"John Doe":25}, {"Mary":30}, "Smith", 20, true]`

数组组成的数组

```
[  
  [1, 2, 3, 4],  
  [2, 3, 4, 5],  
  [5, 2, 3, 5]  
]
```

程序结构：上下文

上下文（静态语义）结构通常在动态执行之前定义语义，例如变量初始化和类型。类型一致性通常通过类型规则来检查。

```
string str = "abc"; int i;
```

```
i = 1 + str;
```

说明和赋值在语法上是正确的，
但赋值语句从上下文来看是不正确的。

编译器可以做什么：

(1)词汇，(2)句法和(3)上下文检查，但不进行语义检查

程序结构：语义

语义描述了程序的含义。C编译器无法检测到语义错误，例如

可以实现为

```
int z;  
z = y/x;
```

```
int x = 0;  
int y = 5;  
int z = y / x;
```

```
int x = 0, y, z;  
x = 3;  
y = 5;  
z = y/(y-x-2);
```

如果缺少说明，C语言会出现什么类型的错误？

- 词汇
- 句法
- 上下文
- 语义

- 命令式和 OO 语言没有正式的定义。
- 函数式编程语言具有基于所使用数学的正式语义定义。
- 在逻辑编程语言中，逻辑表达式通常用于描述语义。
- 语义语言：例如 Prolog、RDF（资源描述框架）和 OWL（Web 本体语言）

- ❑ 在 Python 中没有创建变量的说明
- ❑ 在 Python 中创建变量是通过赋值运算符“=”完成的
- ❑ 赋值发生时
 - 即创建变量
 - 它根据赋值分配数据类型
 - 它被赋予一个值
- ❑ 标识符（变量名）可以以大写或小写字母或“_”字符开头
- ❑ 标识符可以包含字母、数字或下划线
 - 它不能使用其他常见字符，例如 @、\$ 或 %

- ❑ 类（稍后讨论）名称以大写字母开头。
- ❑ 所有其他标识符都以小写字母开头。
 - 以单个前导下划线开头的标识符表示该标识符是私有标识符。
 - 以两个前导下划线开头的标识符表示该标识符是高度私有的标识符。
- ❑ 如果标识符也以两个后置下划线结尾，则标识符是语言定义的特殊名称。
例如：

```
__name__ == '__main__'
```