Selection

Ref Pseudocode

IF, THEN, ELSE, ENDIF

Selection 选择 runs one group of statements or another, depending on a condition 条件.

```
IF <condition>
  THEN
    <statements>
ENDIF
```

```
DECLARE Mark : INTEGER
Mark ← 72
IF Mark >= 50
  THEN
    OUTPUT "Pass"
ENDIF
```

Add `ELSE` for the other case. Exactly one of the two branches 分支 runs — never both, never neither.

```
DECLARE Mark : INTEGER
Mark ← 41
IF Mark >= 50
  THEN
    OUTPUT "Pass"
  ELSE
    OUTPUT "Fail"
ENDIF
```

The layout is the one part students lose marks on, so learn it as a shape:

- `IF` and its condition on one line, **nothing after the condition**.
- `THEN` and `ELSE` on their own lines, indented **two** spaces.
- The statements indented **four**.
- `ENDIF` back at the column of its `IF`.

Nesting 嵌套

An `IF` inside an `IF` handles more than two outcomes. Each one needs its own `ENDIF`.

```
DECLARE Mark : INTEGER
Mark ← 85
IF Mark >= 80
  THEN
    OUTPUT "Distinction"
  ELSE
    IF Mark >= 50
      THEN
        OUTPUT "Pass"
      ELSE
        OUTPUT "Fail"
    ENDIF
ENDIF
```

Common mistakes

- Forgetting `ENDIF`. Every `IF` closes, and a missing one is a syntax error 语法错误 the marker will see at a glance.
- `IF Mark >= 50 THEN OUTPUT "Pass"` on one line. **No such form exists** in either syllabus; write the three lines.
- `IF Mark = 50 THEN` written as `IF Mark ← 50`. The arrow assigns; `=` compares.

CASE OF

When one variable is tested against several **single values**, `CASE OF` says it far more clearly than a stack of nested `IF`s.

```
CASE OF <identifier>
    <value 1> : <statement>
    <value 2> : <statement>
    OTHERWISE <statement>
ENDCASE
```

```
DECLARE Choice : INTEGER
Choice ← 2
CASE OF Choice
    1 : OUTPUT "You chose north"
    2 : OUTPUT "You chose south"
    3 : OUTPUT "You chose east"
    4 : OUTPUT "You chose west"
    OTHERWISE OUTPUT "That is not a direction"
ENDCASE
```

- Each branch is a **value**, a colon, then what to do.
- `OTHERWISE` catches everything the listed values missed. It is optional, and writing one is good practice — it is what happens when the user types something unexpected.
- The first matching branch runs, and the rest are skipped.

A range, and a character

```
DECLARE Grade : CHAR
Grade ← 'B'
CASE OF Grade
    'A' : OUTPUT "Excellent"
    'B' : OUTPUT "Good"
    'C' : OUTPUT "Satisfactory"
    OTHERWISE OUTPUT "Unclassified"
ENDCASE
```

Common mistakes

- Using `CASE OF` for a **condition**. It tests one variable against values; `Mark >= 50` needs an `IF`.
- Forgetting `ENDCASE`.
- Leaving out `OTHERWISE` and then wondering why an unexpected value does nothing at all.