Typology of programming languages

Return Statement

Return Statement

What is the purpose of the return statement?

Is there a best way to return something?

Is there a best way to return something?

Table of Contents

Return via a dedicated keyword 1/2

```c
int compute(int a, int b) {
  int res = a+b;
  // Some computation
  return res;
}
```

C's return statement uses the **return** keyword

```c
int compute(int a, int b) {
  int r_val = a+b;
  // Some computation
  return r_val;
}
```

Java's return statement also uses the **return** keyword

Return via a dedicated keyword 2/2

The **return** statement breaks the current fonction (also for C++, Java, Ada, Modula2).

- Clarity
- Complexify the code
  - No naming convention
  - No homogeneous return inside a given fonction
  - Blur the comprehension via initialisation, intermediate computation, ...

Table of Contents

Return via function's name (1/2)

```
function sum (a, b: integer)
         : integer;
begin
  sum := a + b;
end;
```

Pascal's return statement uses the name of the function

Return via function's name (2/2)

The name of the function is treated as a variable name (also for Fortran, ALGOL, ALGOL68, Simula)

- The "return" may not be the latest statement
- Ambiguous
  - For recursion **sum** denotes a variable **AND** a function
  - Is *somevar* := *sum* legal? (Yes for Pascal, No for Fortan)

Table of Contents

Return via a specific variable (1/2)

```
always_true : BOOLEAN
do
 Result := true
end
```

```
always_one : INTEGER
do
 Result := 1
end
```

```
always_bar : STRING
do
 Result := "bar"
end
```

Effeil's return statement uses the keyword **Result**

Return via a specific variable (2/2)

- The value returned by a function is whatever value is in **Result** when the function ends.
- The return value of a feature is set by assigning it to the **Result** variable (initialised automatically to a default value).
- Unlike other languages, the **return** statement does not exist.

Only in Effeil (to my knowledge)

- Clarity
- Ambiguous if the langage support nested fonctions

Table of Contents

Return the last computed value 1 /2

```
(defun double (x) (* x 2))
```

Lisp's return value is the last computed

```
fn is_divisible_by(lhs: u32,
                   rhs: u32)
                  -> bool {
    if rhs == 0 {
        return false;
    }
    // The `return` keyword
    // isn't necessary
    lhs % rhs == 0
}
```

Same for **Rust's return value**

Return the last computed value 2 /2

In **expression-oriented programming language** (also Lisp, Perl, Javascript and Ruby) the return statement can omitted.

- Instead that the last evaluated expression is the return value.
- A ”last expression” is mandatory in Rust
- If no “return” Python returns **None** and Javascript **undefined**

Table of Contents

Named return values and Naked return

```
func make(r int, i int)
         (re int, im int) {
  re = r
  im = i
  return
}
```

Go combines Named returns values and naked return

Named return values and Naked return

- No declaration/initialisation in the body of the function
- It serves as documentation.

- Functions that return multiple values are hard to name clearly GetUsernameAndPassword
- The signature of the function is slightly more difficult to read

Table of Contents

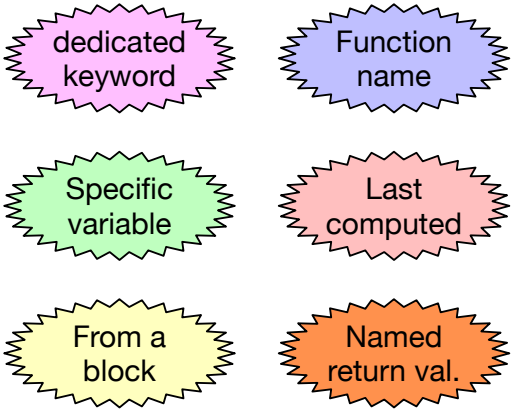
return-from

return-from in lisp

```
(block alpha
    (return-from alpha 1) 2)
```

- Provide a structured lexical non-local exit facility
- Faster than a try-catch (also developped by Lisp with errorset or PL/I)

Summary


dedicated keyword
Function name
Specific variable
Last computed
From a block
Named return val.