More on polymorphism in Chapter 7.

Simplest is ad hoc polymorphism, which really doesn't deserve the name. There are several different variants. Means, literally, "having many forms."

overloading is also known as "ad hoc polymorphism." Some overloading happens in almost all languages.

Lambda expressions

ordinary subroutine invocation.

its methods. The operator() mechanism of C++ makes this look like subroutine closure; pass the object to somebody; let them invoke one of its methods which hold values that would have been in the referencing environment of a subroutine if you don't have formal subroutines.

Note 1: object closures if you don't have formal subroutines.

Example of why deep binding matters for static scope (in C, which lack nested subroutines, or Modula).

Example: (*

first: can pass, return, store

second: can pass, but not return or store

the simplest is code address plus a copy of the static link. A subroutine reference together with its bundled referencing environment is passed or saved, this referencing environment is passed or saved as

for subroutines passed as parameters, this environment captured by deep binding; others (eg. C++, Ada, function return in Pascal)

binding rules are therefore

The original Lisp made this behavior available when desired; it's the overload norm; modern data allocation methods are better

increasingly common in imperative languages as well. Lambda expressions

The equivalent code in Python would:

```
def f(l, k):
    if l == []:
        return
    elif k == l[0]:
        return l[0]
    else:
        return f(l[1:], k)
```

Here the n inside the call inside the lambda captures the "right" h in the nonlocal referencing environment of a

For subroutines passed as parameters, this environment captured by deep binding; others (eg. C++, Ada, function return in Pascal).

Note 2: To the best of my knowledge, no language with static (lexical) local/global references since all local references, that is, those that aren't in the calling environment, must either be passed explicitly or be included in some way in each

Lambda expressions

"binding" is the string we use to indicate that an environment at a particular level of nesting has captured something by value or by reference.

A subroutine reference together with its bundled referencing environment is passed or saved, this referencing environment is passed or saved as

For subroutines passed as parameters, this environment captured by deep binding; others (eg. C++, Ada, function return in Pascal).

Note 1: object closures if you don't have formal subroutines.

Example of why deep binding matters for static scope (in C, which lack nested subroutines, or Modula).

Example: (*

first: can pass, return, store

second: can pass, but not return or store

the simplest is code address plus a copy of the static link. A subroutine reference together with its bundled referencing environment is passed or saved, this referencing environment is passed or saved as

for subroutines passed as parameters, this environment captured by deep binding; others (eg. C++, Ada, function return in Pascal)

binding rules are therefore

The original Lisp made this behavior available when desired; it's the overload norm; modern data allocation methods are better

increasingly common in imperative languages as well. Lambda expressions

The equivalent code in Python would:

```
def f(l, k):
    if l == []:
        return
    elif k == l[0]:
        return l[0]
    else:
        return f(l[1:], k)
```

Here the n inside the call inside the lambda captures the "right" h in the nonlocal referencing environment of a

For subroutines passed as parameters, this environment captured by deep binding; others (eg. C++, Ada, function return in Pascal).

Note 2: To the best of my knowledge, no language with static (lexical) local/global references since all local references, that is, those that aren't in the calling environment, must either be passed explicitly or be included in some way in each

Lambda expressions

"binding" is the string we use to indicate that an environment at a particular level of nesting has captured something by value or by reference.

A subroutine reference together with its bundled referencing environment is passed or saved, this referencing environment is passed or saved as

for subroutines passed as parameters, this environment captured by deep binding; others (eg. C++, Ada, function return in Pascal).

Note 1: object closures if you don't have formal subroutines.

Example of why deep binding matters for static scope (in C, which lack nested subroutines, or Modula).

Example: (*

first: can pass, return, store

second: can pass, but not return or store

the simplest is code address plus a copy of the static link. A subroutine reference together with its bundled referencing environment is passed or saved, this referencing environment is passed or saved as

for subroutines passed as parameters, this environment captured by deep binding; others (eg. C++, Ada, function return in Pascal).

Note 1: object closures if you don't have formal subroutines.

Example of why deep binding matters for static scope (in C, which lack nested subroutines, or Modula).