Binding and Scoping

Binding

- Binding - the operation of associating two things, like a name and the entity it represents. Attributes of parts of programs must be 'bound' to objects before or during computation.

- Binding time - the moment when the binding is performed (compilation, execution, etc).

- Static binding - dynamic binding. Refers to compilation vs. runtime.

Binding Time

- Language design - fundamental aspects of the language, built-in functions, keywords.
- Language implementation - details such as the size of each type, runtime exceptions.
- Programming - algorithms, design of data structures.
- Compilation - mapping between higher-level constructs and machine code, static data.
- Linking - layout of program in memory.
- Load - virtual addresses, dynamic libraries.
- Runtime - virtual functions, values to variables, many more.

Scoping & Rules

- Scope of a binding - the textual region of the program in which a binding is active.

- Scope – the scope of a variable is the range of the program instructions where the variable is known - sometimes a region of a program of maximal size in which no binding changes scope.

Scoping of a Binding

- Usually the scope of a binding is determined statically, meaning at compilation time.
- When a function is called that has a local variable, the binding between the variable name and the instance of the variable local to the call is created.
- Any previous bindings for that same variable name are deactivated in the process (or hidden).
- When the function call ends, the previous binding for the name is restored.

Static (Lexical) vs. Dynamic Scopes

- Static scope - when the scope of a binding is determined during compilation.
    - Current binding - the matching declaration whose block most closely surrounds the point in the program where the name is mentioned.
- Dynamic scope - Bindings that are defined at runtime. They depend on the order in which functions are called.
    - Current binding - the one encountered the most recently during the execution and that has not yet been destroyed.

Dynamic Scope - Lisp

```
(defvar y 3) ; y is global

(defun f (x)
     (+ x y)) ; we think that f uses global y
(f 10) ; => 13

(defun g ()
   (let ((y 4) (a 1))
      (f a)))
(g) ; => 5 ; actually, f does not use global y
```

Static Scope – Python

- **Python appears to have dynamic scope, but, in fact, has static scope.**
- **Example**:

```
pi = 3.1415

def area(r):
        return pi*r*r

area(10) # 314.1500
pi = 3
Area(10) # 300
```

*** in the second call to Area(), pi is changed.**

Static Scope - Python

- **Another example:**

```
pi = 3.1415
pi_holder = 10
def create_area():
        pi_holder = pi # local pi_holder

        def area(r):
                return pi_holder*r*r

        return area
area = create_area()
area(10) # 314.15
pi_holder # Still 10
pi_holder = 3
area(10) # Still 314.15
```

Static Scope - Python

- **Analysis:**
  - The first example works, but is problematic. However, in the second example, when we declare pi_holder=10, which: (1) pi_holder inside of create_area() is a local variable; (2) the local pi_holder is in area‘s scope, thus changing the value of pi_holder does not affect it.

Static Scope - Python

- **A third example (re-write of the second python example):**

```
pi = 3.1415
def create_area(pi = pi):

        def area(r):
                return pi*r*r

        return area
area = create_area()
area(10) #314.15
pi = 3
area(10) #314.15
```

**** Python has static scoping! Python functions inherit the surrounding scope, and so to prevent this, you would need to wrap your function definition inside another function that copies in the values you need into its local variables.**