CS315 HW1 / Yasin Balcancı / 21501109 / Sec:1

Python has static scoping because the variables have values according to the nearest binding. For example:

```
x = 5 # x is assigned to 5 first.
def foo():
	x = 6
	print(x)
def bar():
	def cet():
		x = 4
		print(x)
	x = 7
	foo() #assigns x to 5 and prints it. this is another function.
	cet() #assigns x to 4 and prints it. this is an inner function.
	print(x)
bar() # assigns x to 7. calls foo() and cet() respectively. then prints x.
print(x)
```

---

gives 6, 4, 7 and 5 as output respectively, meaning that even if the value of a variable changes in the borders of another function, it keeps the old value in the borders of the prior function.

JavaScript, on the other hand, has dynamic scoping. For example:

```
function main(){
	var x = 5;
		function foo() {
		x = 6;
		document.getElementById("out1").innerHTML = x;
		}
		function bar(){
		function cet(){
			x = 4;
			document.getElementById("out2").innerHTML = x;
		}
		x = 7;
		foo();
		cet();
		document.getElementById("out3").innerHTML = x;
		}
	bar();
	document.getElementById("out4").innerHTML = x;
}
```

---

code gives the output 6, 4, 4 and 4 respectively. Meaning that when a variable changes, this change is applied to its value independent of the borders.

Also **PHP** has static scoping. The following code gives the output: 6, 4, 7, 5.

```
$x = 5;
function foo(){
        $x = 6;
        echo $x;
        echo "\n";
}
function bar(){
        function cet(){
                $x = 4;
        echo $x;
        echo "\n";
        }
        $x = 7;
        foo();
        cet();
        echo $x;
        echo "\n";
}
bar();
echo $x;
echo "\n";
```

C language has dynamic scoping. The following code gives the output 6, 4, 4, 4.

```
#include <stdio.h>
int main(void){
        int x = 5; // x is assigned to 5 first.
        void foo(){
                x = 6;
                printf("%d\n", x);

        }
        void bar(){
                void cet(){
                        x = 4;
                        printf("%d\n", x);
                }
                x = 7; // x is assigned to 7 in bar and the foo() is called
                foo();
                cet();
```

```
                printf("%d\n", x);
        }
        bar(); // prints 7 and 6 respectively.
        printf("%d\n", x); // prints 6
}
```

Perl language gives the oppurtunity to choose whether a variable will be used with static scoping or dinamic scoping. For example:

```
$x = 5;
sub foo(){
        $x = 6;
        print $x;
        print "\n";
}
sub bar(){
        sub cet(){
                $x = 4;
                print $x;
                print "\n";
        }
        local $x = 7;
        foo(); #assigns x to 5 and prints it. this is another function.
        cet(); #assigns x to 4 and prints it. this is an inner function.
        print $x;
        print "\n";
}
bar(); # assigns x to 6. calls foo() and cet() respectively. then prints x.
print $x;
print "\n";
```

---

code gives the output 6, 4, 4, 5. With the notation “local” we made the variable “x” dynamic, that is why both the 2nd and the 3rd outputs are 4. However, because we user the notation “local” in the borders of function bar, it behaved as static outside it, giving the last output 5. On the other hand, we could use “my” notation instead of “local”, meaning that the variable will have static scoping. For example:

```
$x = 5;
sub foo(){
        $x = 6;
        print $x;
        print "\n";
}
sub bar(){
        sub cet(){
```

```
                $x = 4;
                print $x;
                print "\n";
        }
        my $x = 7;
        foo(); #assigns x to 5 and prints it. this is another function.
        cet(); #assigns x to 4 and prints it. this is an inner function.
        print $x;
        print "\n";
}
bar(); # assigns x to 6. calls foo() and cet() respectively. then prints x.
print $x;
print "\n";
```

---

code gives the output: 6, 4, 7, 4. Because the notation “my” is used in the borders of function bar, it behaved as it has static scoping only in that borders. The last output was 4 because the last time x was changed in a function different from bar, it got the value 4.