Variable Scopes: Takeaways ⇗

Syntax

USING BUILT-IN FUNCTIONS

- To find the the length of an object: `len()`

- To return a float: `float()`

- To find the smallest item: `min()`

- To find the largest item: `max()`

OVERWRITING A BUILT-IN FUNCTION

- To overwrite the built-in function sum():

```
b = [1,2]

sum = sum(b)

sum(20)
```

SCOPES

- Variables defined inside a function are not connected to variables defined outside the function:

```
def add(a,b):
    total = a + b
    return total

total = 15

print(add(10, 20))

print(total)
```

SCOPE ISOLATION

- Variables defined within one function are not connected to variables defined in another function:

```python
def add(a,b):

    total = a + b

    return total

def subtract(a,b):

    total = a - b

    return total

print(add(1,5))

print(subtract(1,5))
```

SCOPE INHERITANCE

- When our code uses a variable name in a *local scope* that it hasn't defined there yet, the Python interpreter will check whether the variable exists in the *global scope*:

```python
total = 50

def find_average(column):

    length = len(column)

    return total / length
```

BUILT-IN INHERITANCE

- If a variable is not located in local or global scope, the interpreter will check the built-in:

```python
def total(a):

    return sum(a)
```

- We define global variables using the global keyword:

```
total = 10

def add_to_total(a):

    global total

    total = total + a

add_to_total(20)

print(total)
```

Concepts

- When we use a variable in python, the interpreter will look for its value according to simple rules:
  - Look for variables within the *local scope*.
  - Look at any *enclosing scopes*, starting with the innermost.
  - Look for variables in the *global scope*.
  - Look in the built-in functions.
  - Throw an error if it doesn't find the variable.

Resources

- Python Documentation on Built-In Functions