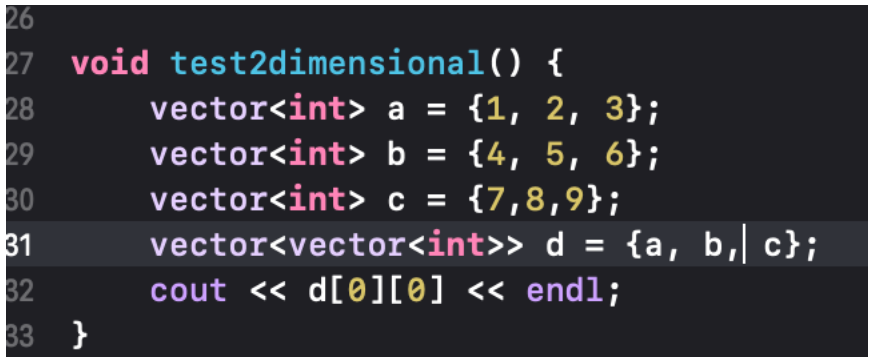
Ch3: Types

by Lina Liu

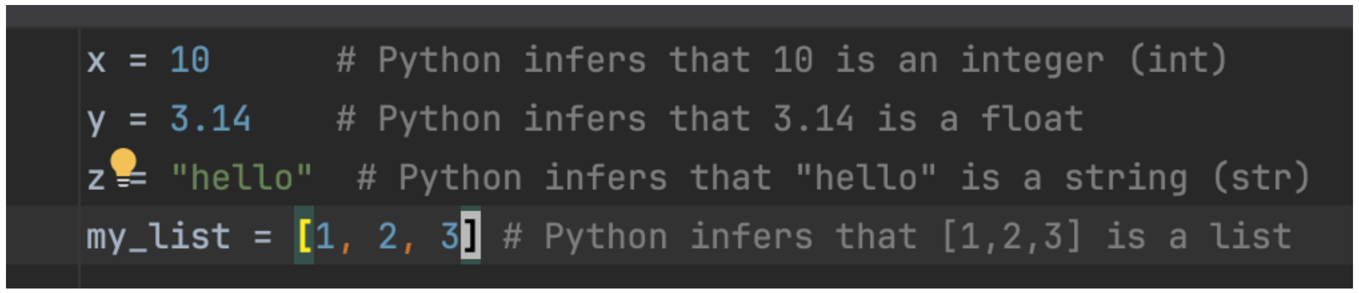
Review of Key Topics

- Python Type
- Create Operations of basic types
- Add Operations of basic types
- Remove Operations of basic types
- f-formatting

Python Type



```cpp
void test2dimensional() {
    vector<int> a = {1, 2, 3};
    vector<int> b = {4, 5, 6};
    vector<int> c = {7,8,9};
    vector<vector<int>> d = {a, b, c};
    cout << d[0][0] << endl;
}
```

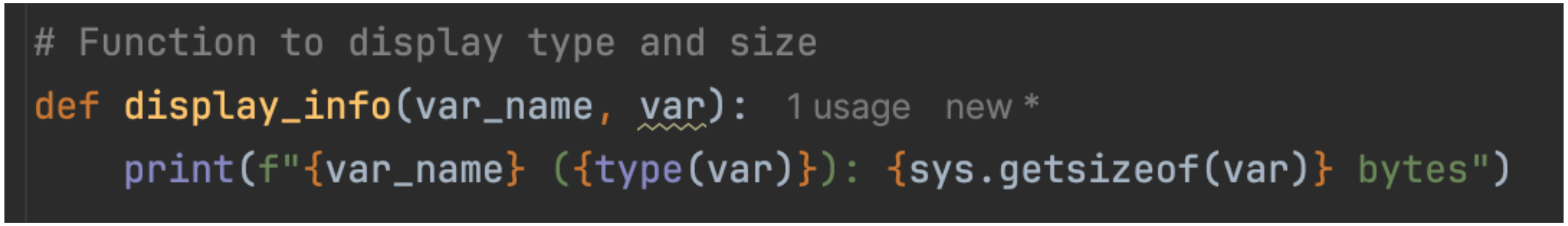


```python
x = 10       # Python infers that 10 is an integer (int)
y = 3.14     # Python infers that 3.14 is a float
z = "hello"  # Python infers that "hello" is a string (str)
my_list = [1, 2, 3] # Python infers that [1,2,3] is a list
```

C++: Static Type

Python: Dynamic Type

Q: As users, how do we know the type and size of a variable?



```python
# Function to display type and size
def display_info(var_name, var):  1 usage  new *
    print(f"{var_name} ({type(var)}): {sys.getsizeof(var)} bytes")
```

Choose Types

- Use List for ordered, mutable sequence of items
  - A playlist of songs
  - A shopping list
- Use Tuple for immutable, ordered sequence of items, often when element position, and not just the relative ordering of elements, is important.
  - RGB Colors
  - latitude and longitude of a landmark
  - Returning Multiple Values from a Function
  - Using as Keys in Dictionaries
- Use Set for unordered collection of unique elements
  - Removing Duplicates
  - Mathematical Set
- Use Dictionary for describe associative relationships.
  - word vocabulary
  - Person’s infomation

Create

- string
  - string_variable = "Hello, Python!"
  - string_variable = ‘Hello, Python!’
- List
  - test_list = list([1,2,3])
  - list_of_mixed_types = [1, "hello", 3.14, True]
- Tuple
  - test_tuple = tuple([1,2,3])
  - tuple_of_mixed_types = (1, "hello", 3.14, True)
- Dictionary
  - my_dict = dict(a=1, b=2, c=3) # Keys become strings
  - person = {"name": "Alice", "age": 30, "city": "New York"}
- Set
  - set1 = set([1, 2, 2, 3, 4, 4, 5])
  - set2 = {1, 2, 2, 3, 4, 4, 5}

Add

- immutable
  - String
    - result = string1 + string2 #Concatenation
  - Tuple
    - new_tuple = tuple1 + tuple2
- mutable
  - List
    - my_list.append(4)
    - my_list.insert(1, 10) # Insert 10 at index 1
    - my_list.extend(another_list)
    - new_list = list1 + list2 #Concatenation
  - Dictionary
    - my_dict["c"] = 3 # Adds "c": 3 to the dictionary
    - my_dict.update(other_dict)
    - my_dict |= new_dict # combines both dictionaries
    - ERROR: dict3 = dict1 + dict2
  - Set
    - my_set.add(4)
    - my_set.update([5, 6, 7]) # Adds 5, 6, and 7
    - new_set = my_set.union(other_set)
    - my_set |= other_set # my_set is updated
    - ERROR: set3 = set1 + set2

Remove

- immutable
  - String
    - Strings are immutable, only create *new* strings.
      - stripped_chars = my_string_2.strip("*") # removes * from both ends
      - replaced_string = my_string_3.replace("test", “example")
      - removed_range = my_string_4[2:7]
  - Tuple
    - Tuples are immutable, only create *new* tuples.
- mutable
  - List
    - my_list.remove(item)
    - my_list.pop(index)
    - del my_list[1:3] # Removes elements from index 1 up to (but not including) index 3
    - my_list.clear() # Removes all elements from the list
  - Dictionary
    - my_dict.pop(key)
    - del my_dict[key]
    - my_dict.clear() # Removes all items
    - my_dict.popitem() # Removes and returns the *last* inserted item as a tuple (key, value)
  - Set
    - my_set.remove(item)
    - my_set.discard(item)
    - my_set.pop() # removes and returns a *random* element
    - my_set.clear()

f-strings

- Basic Usage: Embedding variables
- Expressions inside f-strings
- Format Specifications: Controlling output
- Debugging with = sign — Prints both the expression and the result