# 1.2 The Python Memory Model: Functions and Parameters

## HOW FUNCTION CALLS ARE TRACKED

- Stack Frame: keeps track of the function its running and the variables defined inside of it.
- Call Stack: pile of frames that already exist

**FUNCTION CALLS:**

1. New frame is created and placed on top of call stack
2. Each parameter is defined inside frame
3. Argument in function call is evaluated & object is assigned to corresponding parameter.
4. Body of function is executed & local variables are stored in the frame

**FUNCTION RETURNS:**

- Executing a return statement / end of the function
- Frame of function call is deleted : parameters & local variables disappear

**PASSING AN ARGUMENT CREATES AN ALIAS:**

- "Parameter passing" -> variable assignment
- If argument to function is a variable :
  - The ID of the object the variable references is assigned to function's parameters (THE ALIAS)

**PASSING REFERENCE TO AN IMMUTABLE OBJECT:**

- No effect outside the function

**PASSING REFERENCE TO A MUTABLE OBJECT:**

- Has an effect

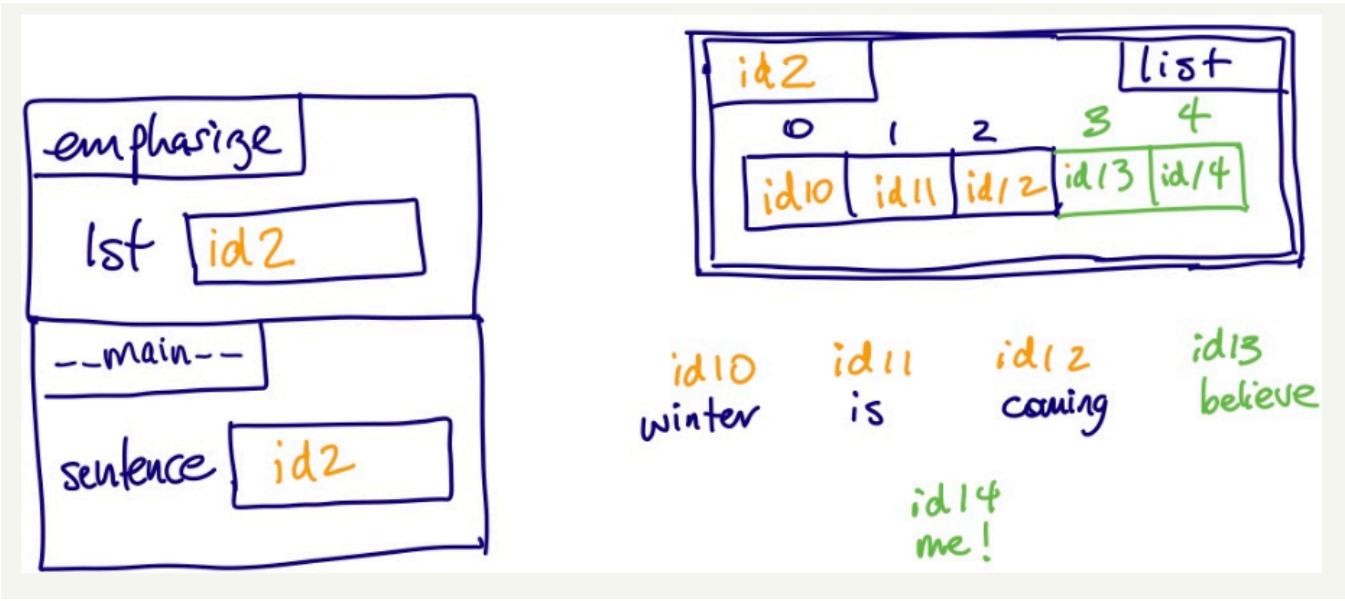
```
def emphasize(lst: List[str]) -> None:
    lst = lst + ['believe', 'me!']

if __name__ == '__main__':
    sentence = ['winter', 'is', 'coming']
    emphasize(sentence)
    print(sentence)
```

- The parameter `lst` exists in the stack frame. It comes into being when the function is called. And when the function returns, this frame will be discarded, along with everything in it. At that point, `lst` no longer exists.

```
def emphasize(lst: List[str]) -> None:
    lst.extend(['believe', 'me!'])

if __name__ == '__main__':
    sentence = ['winter', 'is', 'coming']
    emphasize(sentence)
    print(sentence)
```




- When we pass argument `sentence` to `lst`, we assign it to `lst`. In other words, we set `lst` to `id2`, which creates an alias.
- `id2` is a reference to a `list` object, which is mutable. When we use `lst` to access and change that object, the object that `sentence` references also changed. Of course it does: they are the same object!