

# Simple Stack Frame Examples

CS21 at Swarthmore College

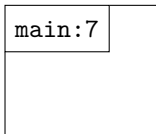
## Basic Example

```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```

At the beginning of the program, `main` is called. We create a new stack frame. Since `main` has no parameters, the stack frame is empty.

# Basic Example

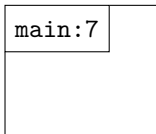
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



At the beginning of the program, `main` is called. We create a new stack frame. Since `main` has no parameters, the stack frame is empty.

## Basic Example

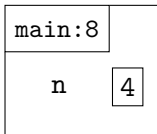
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



When line 7 of `main` is executed, the variable `n` is set to the value 4. We signify this by drawing a box in the stack frame and labeling it with the variable name. We put the contents of the variable in the box.

## Basic Example

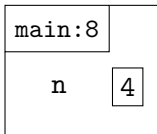
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



When line 7 of `main` is executed, the variable `n` is set to the value 4. We signify this by drawing a box in the stack frame and labeling it with the variable name. We put the contents of the variable in the box.

## Basic Example

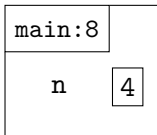
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



When line 8 is executed, we will call `f`. To do so, we must first determine the value of each of its arguments. In this case, the first parameter is `n`, whose value is currently 4. The second parameter is just 2.

# Basic Example

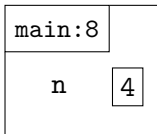
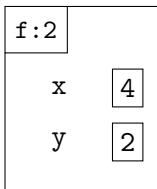
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



Once we've established the value of the arguments on line 8 (4 and 2, respectively), the `f` function is called. We create a new stack frame. Since `f` has two parameters, we create variables for them in the stack frame. They contain the values of their corresponding arguments.

## Basic Example

```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```

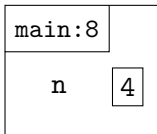
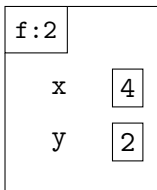


Once we've established the value of the arguments on line 8 (4 and 2, respectively), the `f` function is called. We create a new stack frame. Since `f` has two parameters, we create variables for them in the stack frame. They contain the values of their corresponding arguments.



## Basic Example

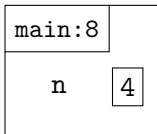
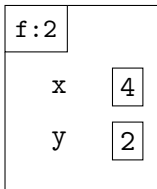
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



Note that the stack frame for main is keeping track of where we were in that function. When we are done with f, we will return to that line.

## Basic Example

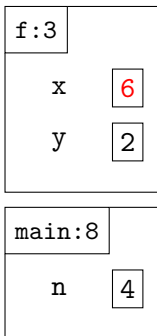
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



When we run line 2 in `f`, we will update the variable `x` by adding the contents of the variable `y` to it. We change the stack diagram accordingly.

## Basic Example

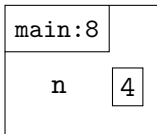
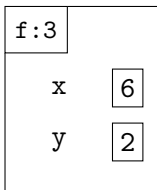
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



When we run line 2 in `f`, we will update the variable `x` by adding the contents of the variable `y` to it. We change the stack diagram accordingly.

## Basic Example

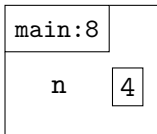
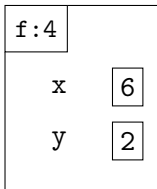
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



Line 3 will print the contents of the x variable: in this case, 6.

## Basic Example

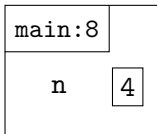
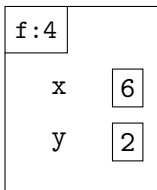
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



Line 3 will print the contents of the x variable: in this case, 6.

## Basic Example

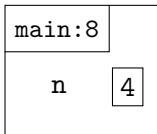
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



Line 4 will return the value of `x` to the place where `f` was called. As a result, the variable `out` in `main` is given the value 6.

## Basic Example

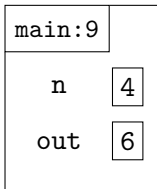
```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



Line 4 will return the value of `x` to the place where `f` was called. As a result, the variable `out` in `main` is given the value 6.

## Basic Example

```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```

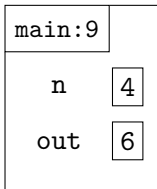


Line 4 will return the value of `x` to the place where `f` was called. As a result, the variable `out` in `main` is given the value 6.



# Basic Example

```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```



Line 9 prints the contents of the out variable (here, 6). After it runs, the main function is complete and the program is finished.

## Basic Example

```
1 def f(x,y):
2     x += y
3     print x
4     return x
5
6 def main():
7     n = 4
8     out = f(n,2)
9     print out
10
11 main()
```

Line 9 prints the contents of the out variable (here, 6). After it runs, the main function is complete and the program is finished.

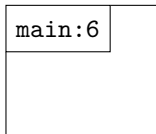
## Lists Example

```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```

As before, `main` is called at the start of this program. We create a new stack frame for it.

## Lists Example

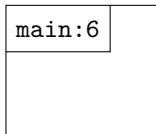
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



As before, `main` is called at the start of this program. We create a new stack frame for it.

## Lists Example

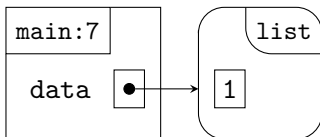
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Line 6 of `main` creates a new list containing just the value 1. A *reference* to that list is stored in the `data` variable. We represent the list by using a rounded box; we represent the reference as an arrow.

## Lists Example

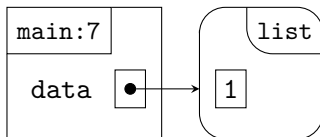
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Line 6 of `main` creates a new list containing just the value 1. A *reference* to that list is stored in the `data` variable. We represent the list by using a rounded box; we represent the reference as an arrow.

## Lists Example

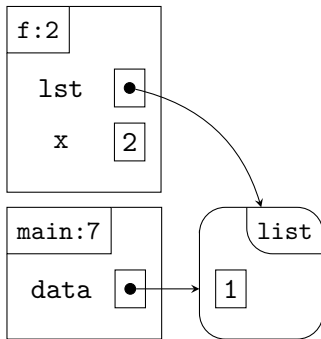
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Line 7 of `main` is a function call. Just as before, we create a new stack frame and copy each argument into its corresponding parameter. Here, we copy the value 2 into the variable `x` and we copy the *reference* from `data` into the variable `lst`.

## Lists Example

```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```

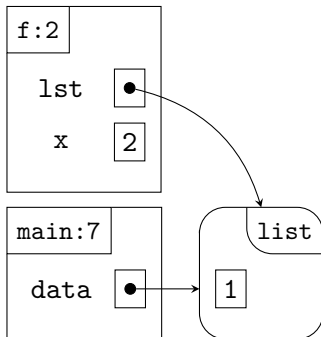


Line 7 of main is a function call. Just as before, we create a new stack frame and copy each argument into its corresponding parameter. Here, we copy the value 2 into the variable x and we copy the *reference* from data into the variable lst.



## Lists Example

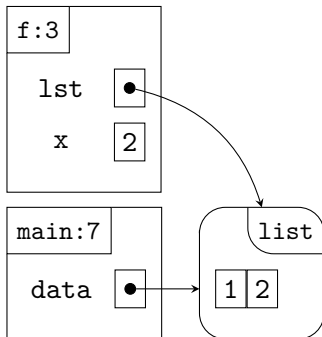
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Line 2 of `add_twice` appends a copy of the value in `x` to the end of the list. Here, that value is 2. We change the list object in our diagram to reflect this.

## Lists Example

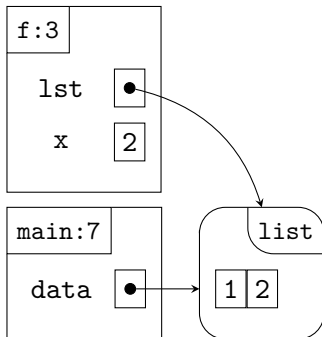
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Line 2 of `add_twice` appends a copy of the value in `x` to the end of the list. Here, that value is 2. We change the list object in our diagram to reflect this.

## Lists Example

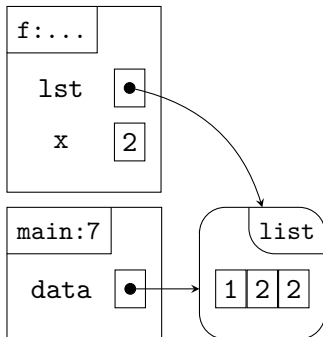
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Of course, line 3 does the same thing; this adds another 2 to our list. Note that this function doesn't return anything; it just adds to the list.

## Lists Example

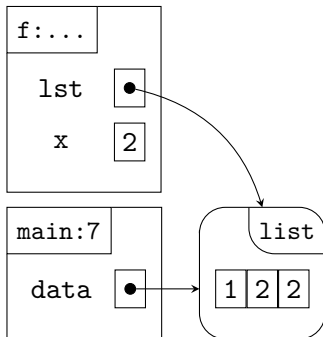
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Of course, line 3 does the same thing; this adds another 2 to our list. Note that this function doesn't return anything; it just adds to the list.

## Lists Example

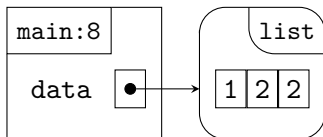
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Once we're finished with the `add_twice` function, we destroy its stack frame and return to executing `main`.

## Lists Example

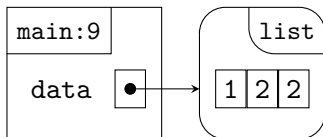
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Line 8 of main prints the contents of the list to which data refers. Because of the call to add\_twice, this list changed. So main prints “[1,2,2]”.

## Lists Example

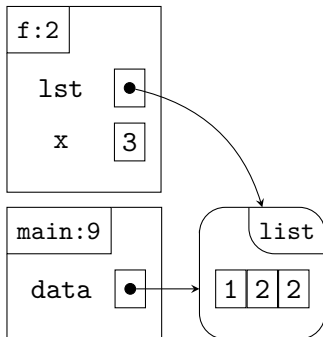
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Line 9 of `main` calls `add_twice` again. Just as last time, we copy the arguments into their respective parameters. This time, `x` is set to 3; `lst` is still set to the same reference as `data`.

## Lists Example

```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```

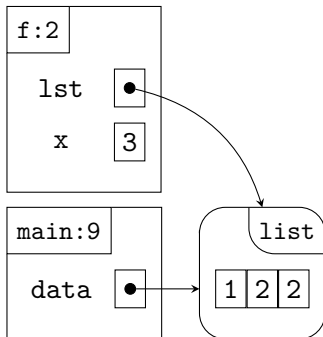


Line 9 of `main` calls `add_twice` again. Just as last time, we copy the arguments into their respective parameters. This time, `x` is set to 3; `lst` is still set to the same reference as `data`.



## Lists Example

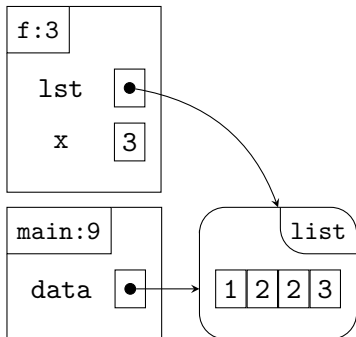
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Once again, `add_twice` adds the value contained in `x` to the list referenced by `lst`; it does this twice.

## Lists Example

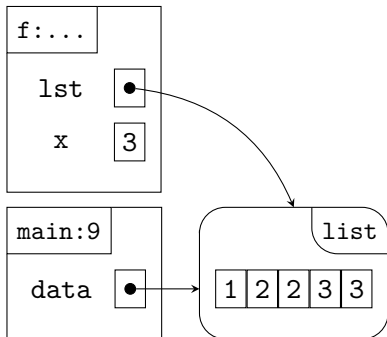
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Once again, `add_twice` adds the value contained in `x` to the list referenced by `lst`; it does this twice.

## Lists Example

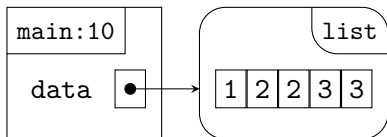
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



Once again, `add_twice` adds the value contained in `x` to the list referenced by `lst`; it does this twice.

## Lists Example

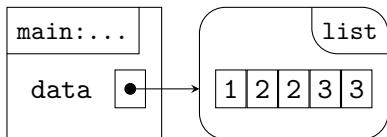
```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



We finish `add_twice`, discarding its stack frame. We return to `main`, where line 10 prints the contents of the list. Because it has been changed again, we print `[1,2,2,3,3]` this time.

## Lists Example

```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```



With that, the program is finished.

## Lists Example

```
1 def add_twice(lst,x):
2     lst.append(x)
3     lst.append(x)
4
5 def main():
6     data = [1]
7     add_twice(data,2)
8     print data
9     add_twice(data,3)
10    print data
11
12 main()
```

With that, the program is finished.