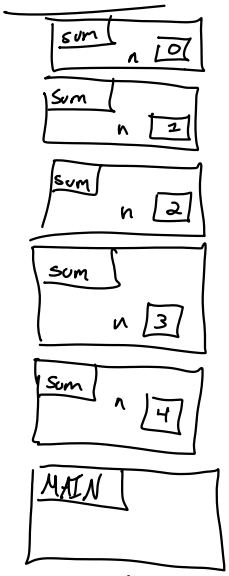
Gull testing

- Test programs w/ tuples w/ different values
- Test programs w/ tuples of different sizes
- Test programs w/ closures
- Test programs w/ cycles
- Test programs which nest tuples

```
let x = (1, (2, (3, 4))) in
let _ = cycle_tuple_memory 8 in
print(x)
```

Recursion



sum.py : sum(4)

```
let sum n =
    if n=0 then 0 else sum(n-1) +n
;;
sum 4
```

```
let sum n =
    if n=0 then 0 else sum (n-1) +n
;;
if 4=0 then 0 else sum(4-1)+4
```

```
let sum n =
    if n=0 then 0 else sum (n-1) +n
;;
sum(4-1)+4
```

```
let sum n =
    if n=0 then 0 else sum (n-1) +n
;;
if 3=0 then 0 else sum (3-1)+3 +4
```

sum.ml

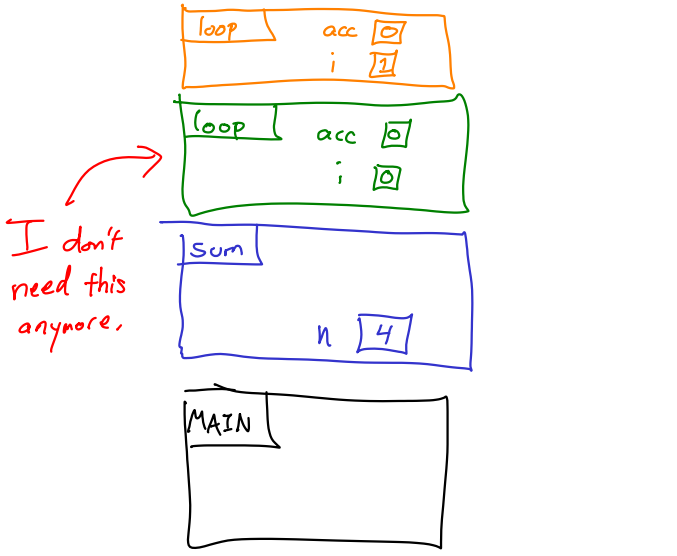
```
let sum n =
    let rec loop acc i =
      if i>n then acc else loop (acc+i) (i+1)
    in
    loop 0 0
;;
sum 4
```

```
let rec loop acc i =
    if i>4 then acc else loop (acc+i) (i+1)
in
loop 0 0
```

```
let rec loop acc i =
    if i>4 then acc else loop (acc+i) (i+1)
in
if 0>4 then 0 else loop (0+0) (0+1)
```

```
let rec loop acc i =
    if i>4 then acc else loop (acc+i) (i+1)
in
loop 0 1
```

```
let rec loop acc i =
    if i>4 then acc else loop (acc+i) (i+1)
in
if 1>4 then 0 else loop (0+1) (1+1)
```



g 0 — can eliminate calling frame

g 0 + 1

1 + g 0 — can't eliminate calling frame

(f 0, g 0)

if f 0 then g 0 else h 0

let x = f 0 in g 0

let a = 4 in
let b = f 0 in
let c = g 0 in
h a c

Tail Call Optimization

part of an expression which happens last

```
type expr =
  | EInt of int
  | EBool of bool
  | EVar of string
  | EUnaryOp of unop * expr
  | EBinaryOp of binop * expr * expr
  | ELet of string * expr * expr
  | EAppl of expr * expr
  | EIf of expr * expr * expr
  | ETuple of expr list
  | ESet of expr * expr * expr
```