“Being a tail call’ is a property of a function call expression.

Every function call is either a tail call, or isn't.

A tail call is **not** a property of a function definition.

```
def f(n):
  n * 2

def g(n):
  f(n + 1)

def h(n):
  f(n + 1) * 3
```

Tail call

Not a tail call

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

A: Yes tail call B: Not tail call

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

A: Yes tail call B: Not tail call

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

A: Yes tail call B: Not tail call

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

```
def f(n):
  n * 2

def g(n):
  f(f(5))
```

A: Yes tail call B: Not tail call

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

```
def f(n):
  n * 2

def g(n):
  f(f(5))
```

A: Yes tail call B: Not tail call

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

```
def dfs(t, k):
  if t[0] == k:
    true
  else: if dfs(t[1], k):
    true
  else:
    dfs(t[2], k)
```

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

```
def f(n):
  n * 2

def g(n):
  f(f(5))
```

A: Yes tail call

B: Not tail call

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

```
def dfs(t, k):
  if t[0] == k:
    true
  else: if dfs(t[1], k):
    true
  else:
    dfs(t[2], k)
```

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

```
type expr =
  | ENumber of int
  | EBool of bool
  | ELet of string * expr * expr
  | EIf of expr * expr * expr
```

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

A | EIf of NO * NO * YES

B | EIf of YES * NO * NO

C | EIf of NO * YES * YES

```
def f(n):
  n * 2

def g(n):
  f(f(5))
```

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

```
type expr =
  | ENumber of int
  | EBool of bool
  | ELet of string * expr * expr
  | EIf of expr * expr * expr
  | EApp of string * expr list
```

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

A | EApp of string * YES

B | EApp of string * NO

```
def f(n):
  n * 2

def g(n):
  f(f(5))
```

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

```
def f(n):
  n * 2

def g(n):
  f(f(5))
```

```
type expr =
  | ENumber of int
  | EBool of bool
  | ELet of string * expr * expr
  | EIf of expr * expr * expr
  | EApp of string * expr list
  | EPrim2 of prim2 * expr * expr
  | EPrim1 of prim1 * expr
  | EPair of expr * expr
```

```
def f(n):
  n * 2

def g(n):
  let x = f(n + 1) in
  f(x + 1)
```

```
def f(n):
  n > 2

def g(n):
  if f(n):
    f(n + 1)
  else:
    5
```

```
def f(n):
  n * 2

def g(n):
  f(f(5))
```

```
type expr =
  | ENumber of int
  | EBool of bool
  | ELet of string * expr * expr
  | EIf of expr * expr * expr
  | EApp of string * expr list
  | EPrim2 of prim2 * expr * expr
  | EPrim1 of prim1 * expr
  | EPair of expr * expr
```