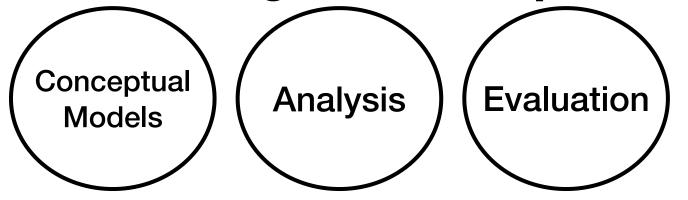
Core Design Concepts Discussed:



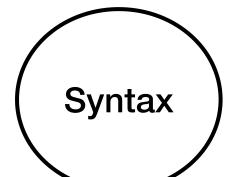
Recursive Functions and their Evaluation

Harley Eades III

Recursive Functions in OCaml (syntax)

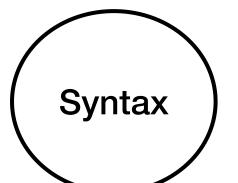
```
# let rec f x = e;
```

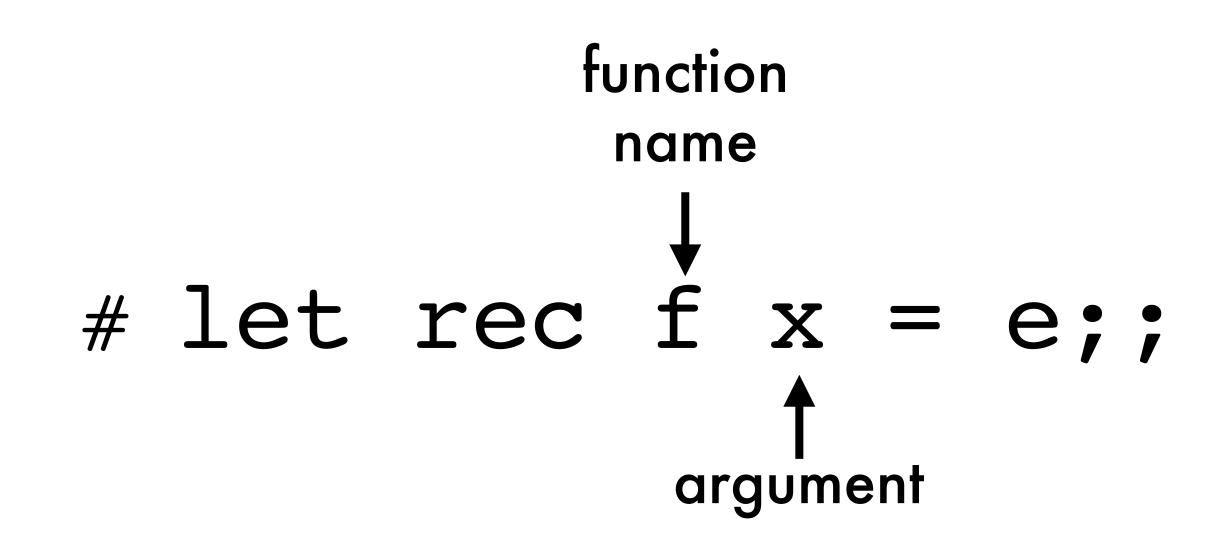
Recursive Functions in OCaml (syntax)



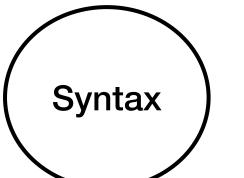
```
function
            name
# let rec f x = e;;
```

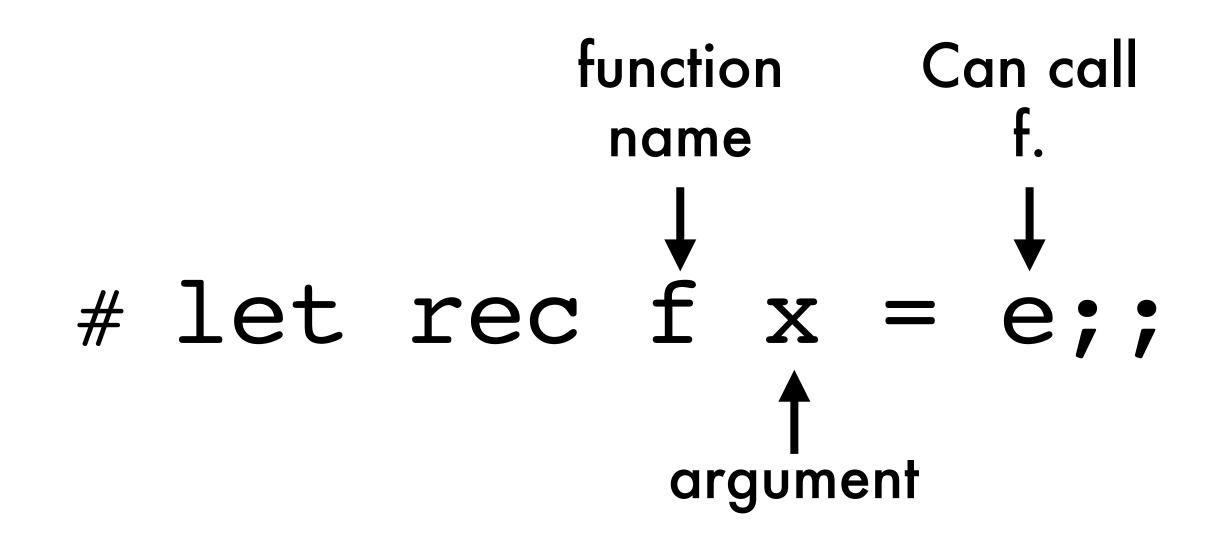
Recursive Functions in OCaml (syntax





Recursive Functions in OCaml (Syntax





Evaluation

But, what about performance?

Activation Record:

The location in memory where an executing function stores its bindings.

Activation records are sometimes referred to as <u>frames</u>.

Consider evaluating the following function:

```
1: let cube n =
2: let c = n*n*n in
3: c;;
4: let main =
5: let n = 5 in
6: let ans = cube n in
7: ans;;
8: main;;
```



Activation Record:

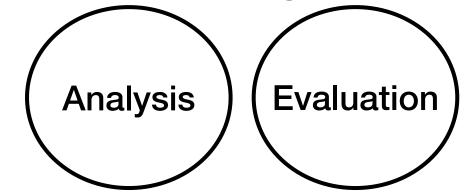
The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as <u>frames</u>.

Consider evaluating the following function:

```
1: let cube n =
2: let c = n*n*n in
3: c;;
4: let main =
5: let n = 5 in
6: let ans = cube n in
7: ans;;
8: main;;
```

Core Design Concepts:



Activation Record: program initialization

Frame	Symbol	Value
init Iine: 8	cube main	<fun><fun></fun></fun>

Activation Record:

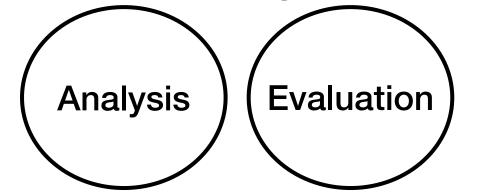
The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as <u>frames</u>.

Consider evaluating the following function:

```
1: let cube n =
2: let c = n*n*n in
3: c;;
4: let main =
5: let n = 5 in
6: let ans = cube n in
7: ans;;
8: main;;
```

Core Design Concepts:



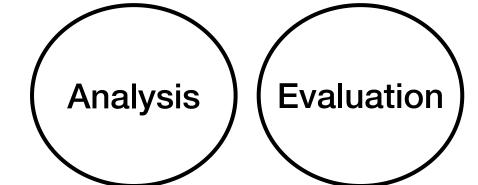
Activation Record: program initialization

Frame	Symbol	Value
init Iine: 8	cube main	<fun><fun></fun></fun>

Activation Record: after calling main

Frame	Symbol	Value
init Iine: 8	cube main	<fun><fun></fun></fun>
main line: 6	n ans	5?

Core Design Concepts:



Activation Record:

The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as <u>frames</u>.

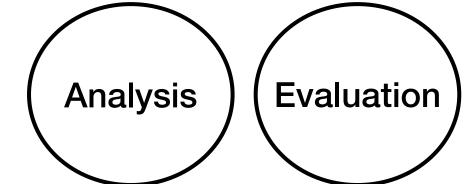
Consider evaluating the following function:

```
1: let cube n =
2: let c = n*n*n in
3: c;;
4: let main =
5: let n = 5 in
6: let ans = cube n in
7: ans;;
8: main;;
```

Activation Record: after calling main

Frame	Symbol	Value
init Iine: 8	cube main	<fun><fun></fun></fun>
main	n	5
line: 6	ans	?
cube	n	5
line: 2	c	125 ←

Core Design Concepts:



Activation Record:

The location in memory where an executing function stores its binding.

Activation records are sometimes referred to as <u>frames</u>.

Consider evaluating the following function:

```
1: let cube n =
2: let c = n*n*n in
3: c;;
4: let main =
5: let n = 5 in
6: let ans = cube n in
7: ans;;
8: main;;
```

Activation Record: after calling main

Frame	Symbol	Value
init Iine: 8	cube main	<fun><fun></fun></fun>
main	n	5
line: 6	ans	125 ←
cube	n	5
line: 2	C	125 ←

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
6.
         else let ack1 = ackermann m (m - 1) in
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
114.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Core Design Concepts:

Evaluation

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
6.
         else let ack1 = ackermann m (m - 1) in
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
114.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```



Frame	Symbol	Value
init line: 17	ackermann main	<fun><fun></fun></fun>

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
         then let ack = ackermann (m - 1) 1 in ack
5.
6.
         else let ack1 = ackermann m (m - 1) in
7.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```



Frame	Symbol	Value
init Iine: 17	ackermann main	<fun><fun></fun></fun>
	m	1
main	n	0
line: 14	ans1	?
	ans2	?

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
6.
         else let ack1 = ackermann m (m - 1) in
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```



Frame	Symbol	Value
init Iine: 17	ackermann main	<fun><fun></fun></fun>
main line: 14	m n ans1 ans2	1 0 ?
ackermann line: 5	m n ack	1 0 ?

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```



Frame	Symbol	Value
init Iine: 17	ackermann main	<fun><fun></fun></fun>
main line: 14	m n ans1 ans2	1 0 ?
ackermann line: 5	m n ack	1 0 ?
ackermann line: 3	m n ret	0 1 2

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
6.
         else let ack1 = ackermann m (m - 1) in
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```



Frame	Symbol	Value
init Iine: 17	ackermann main	<fun><fun></fun></fun>
main line: 14	m n ans1 ans2	1 0 ?
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```



Frame	Symbol	Value
init Iine: 17	ackermann main	<fun><fun></fun></fun>
main line: 14	m n ans1 ans2	1 0 2 ?
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
6.
         else let ack1 = ackermann m (m - 1) in
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
      let n = 0 in
13.
         let ans1 = ackermann m n in
114.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```



Frame	Symbol	Value
	m	1
main	n	0
line: 14	ans1	2
	ans2	?
	m	1
ackermann	n	1
line: 7	ack1	?
	ack2	3

Evaluating Recursive Functions

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
6.
         else let ack1 = ackermann m (m - 1) in
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
111.
      let m = 1 in
12.
      let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
	m	1
main	n	0
line: 14	ans1	2
	ans2	?
	m	1
ackermann	n	1
line: 7	ack1	?

ack2

n

ack

ackermann

line: 5

Evaluation

Evaluation

Evaluating Recursive Functions

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
	m	1
main	n	0
line: 14	ans1	2
	ans2	?
	m	1
ackermann line: 7	n	1
	ack1	?
	ack2	?
ackermann line: 5	~	1
	m	0
	n 1-	?
	ack	
ackermann	m	0
	n	
line: 3		

ret

Evaluation

Evaluating Recursive Functions

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
	m	1
main	n	0
line: 14	ans1	2
	ans2	?
ackermann	m	1
	n	1
line: 7	ack1	?
	ack2	?
ackermann		1
	m 	0
line: 5	n	2
	ack	
	m	0
ackermann		1

n

ret

line: 3

Evaluating Recursive Functions

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
7.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Evaluation

Frame	Symbol	Value
	m	1
main	n	0
line: 14	ans1	2
	ans2	?
	m	1
ackermann	n	1
line: 7	ack1	2
	ack2	?
ackermann	**	1
	m	0
line: 5	n o al-	2
	ack	
ackermann	m	0
line: 3	n	1
iii ie. 3	ret	2

Evaluation

Evaluating Recursive Functions

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
	m	1
main	n	0
line: 14	ans1	2
	ans2	3
	m	1
ackermann	n	1
line: 7	ack1	2
	ack2	?
		1
ackermann	m	0
line: 5	n	2
	ack	
	m	0
ackermann	n	1
line: 3	ret	2
ookormonn	m	0
ackermann	n	2
line: 3	rot	2

ret

Evaluation

Evaluating Recursive Functions

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
	m	1
main	n	0
line: 14	ans1	2
	ans2	?
	m	1
ackermann	n	1
line: 7	ack1	2
	ack2	3
	m	1
ackermann	m	0
line: 5	n	2
	ack	
	m	0
ackermann	n	1
line: 3	ret	2
	m	n

Ш

n

ret

ackermann

line: 3

Evaluation

Evaluating Recursive Functions

Consider evaluating the following recursive function:

```
1. let rec ackermann m n =
   if m == 0
    then let ret = n + 1 in ret
    else if n == 0
5.
         then let ack = ackermann (m - 1) 1 in ack
         else let ack1 = ackermann m (m - 1) in
6.
              let ack2 = ackermann (m - 1) ack1 in
8.
                ack2
9.
10. let main =
11.
      let m = 1 in
12.
       let n = 0 in
13.
         let ans1 = ackermann m n in
14.
           let ans2 = ackermann m m in
15.
             ans1 + ans2
16.
17. main;;
```

Frame	Symbol	Value
main line: 14	m n ans1 ans2	1 0 2 3
ackermann line: 7	m n ack1 ack2	1 1 2 3
ackermann line: 5	m n ack	1 0 2
ackermann line: 3	m n ret	0 1 2
ackermann	m	0

n

ret

line: 3