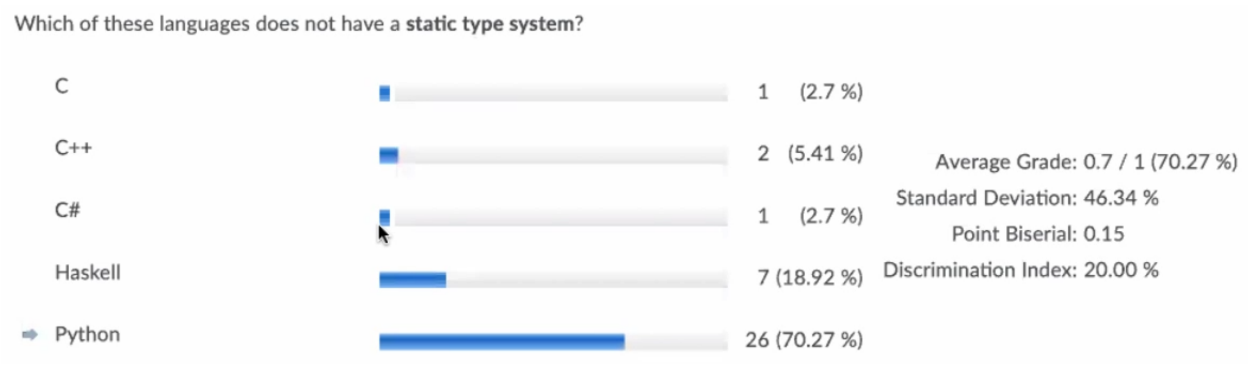
MIDTERM 2

1. Which of these languages does not have a static type system?

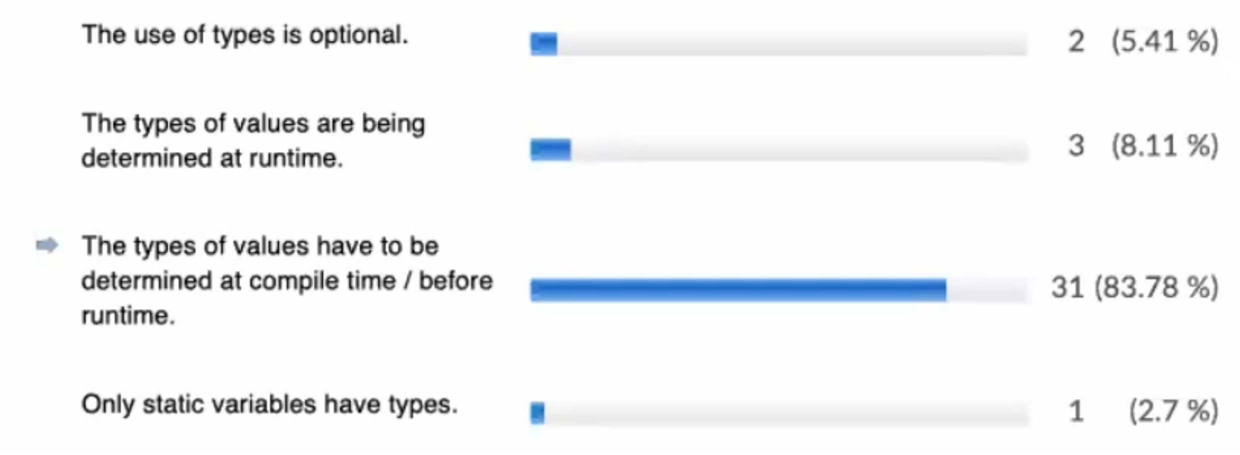


Which of these languages does not have a **static type system**?

C	1	(2.7 %)
C++	2	(5.41 %)
C#	1	(2.7 %)
Haskell	7	(18.92 %)
➡ Python	26	(70.27 %)

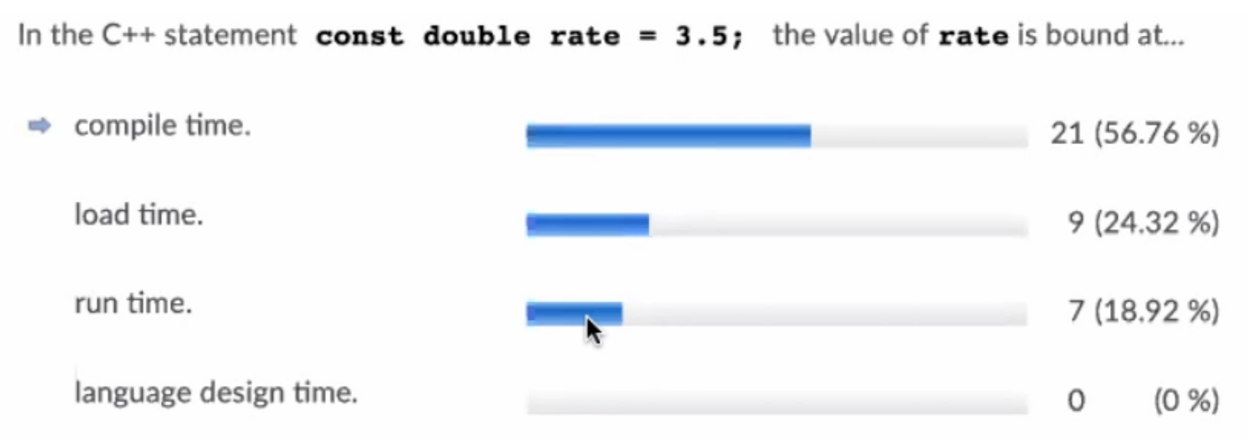
Average Grade: 0.7 / 1 (70.27 %)
Standard Deviation: 46.34 %
Point Biserial: 0.15
Discrimination Index: 20.00 %

2. Match the genetic programming features to the languages.
   a. Mainly just workarounds with macros or void
      i. C
   b. Templates
      i. C++
   c. Type parameters and interfaces
      i. C#
   d. Polymorphic types and type classes
      i. Haskell
3. What characterizes a static type system?



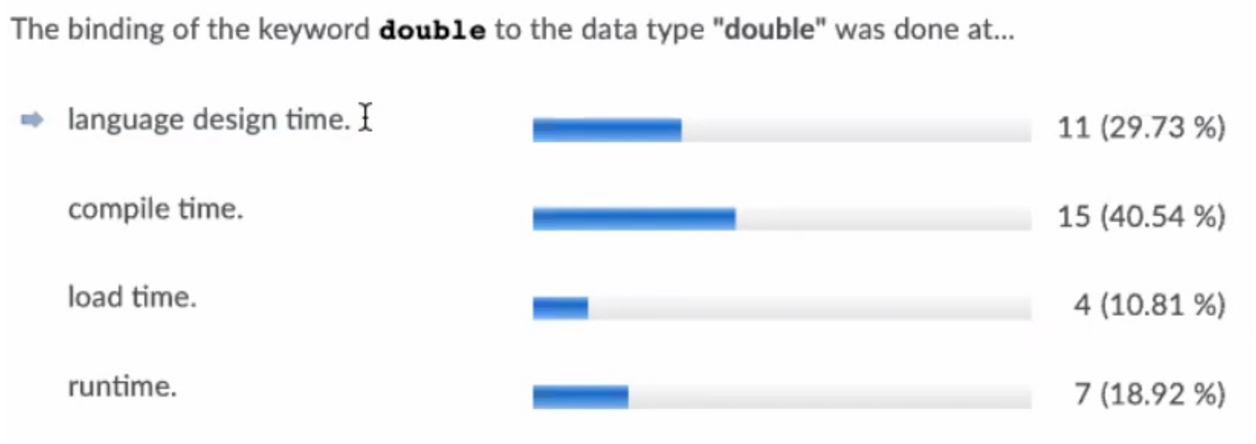
The use of types is optional.	2	(5.41 %)
The types of values are being determined at runtime.	3	(8.11 %)
➡ The types of values have to be determined at compile time / before runtime.	31	(83.78 %)
Only static variables have types.	1	(2.7 %)

4. In the C++ statement const double rate = 3.5; the value rate is bound at…


In the C++ statement **`const double rate = 3.5;`** the value of **`rate`** is bound at...

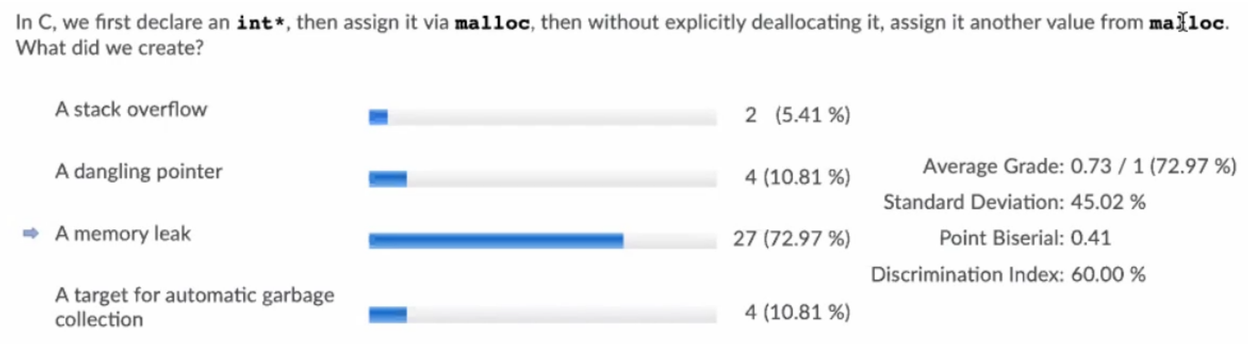
➡ compile time.	21 (56.76 %)
load time.	9 (24.32 %)
run time.	7 (18.92 %)
language design time.	0 (0 %)

5. The binding of the keyword double to the data type "double" was done at...


The binding of the keyword **`double`** to the data type **"double"** was done at...

➡ language design time.	11 (29.73 %)
compile time.	15 (40.54 %)
load time.	4 (10.81 %)
runtime.	7 (18.92 %)

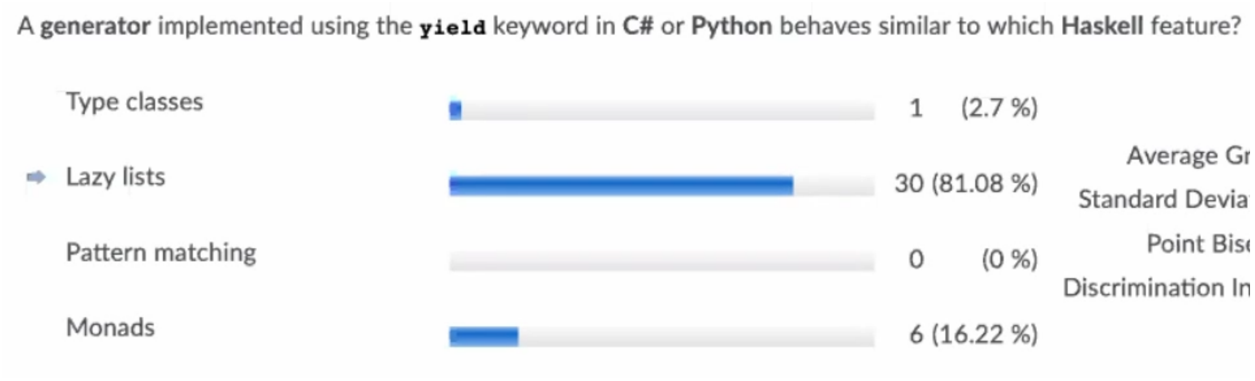
6. In C, we first declare an int*, then assign it via malloc, then without explicitly deallocating it, assign it another value from malloc. What did we create?


In C, we first declare an **`int*`**, then assign it via **`malloc`**, then without explicitly deallocating it, assign it another value from **`malloc`**. What did we create?

A stack overflow	2 (5.41 %)
A dangling pointer	4 (10.81 %)
➡ A memory leak	27 (72.97 %)
A target for automatic garbage collection	4 (10.81 %)

Average Grade: 0.73 / 1 (72.97 %)
Standard Deviation: 45.02 %
Point Biserial: 0.41
Discrimination Index: 60.00 %

7. A generator implemented using the yield keyword in C# or python behaves similar to which Haskell feature?

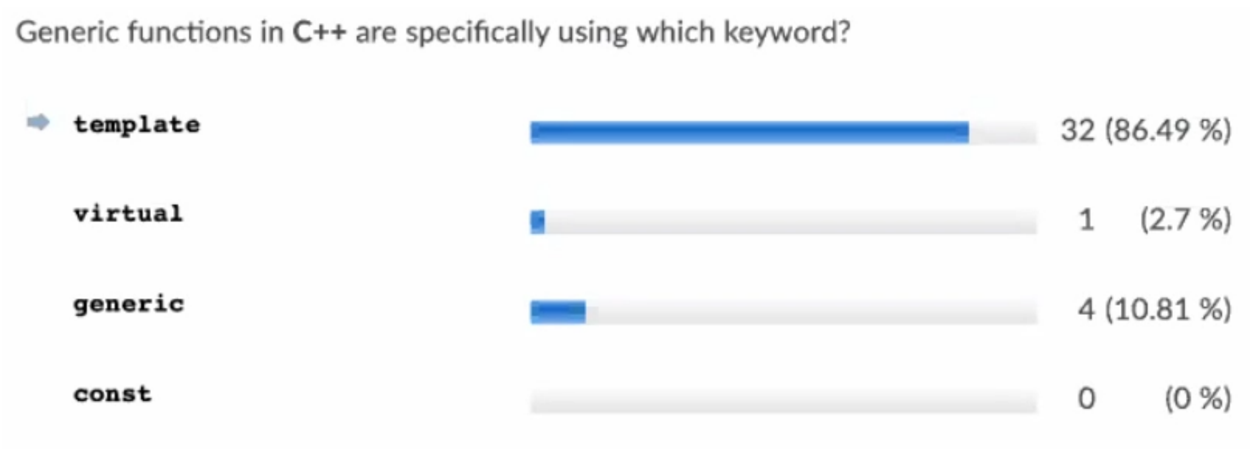


A **generator** implemented using the `yield` keyword in **C#** or **Python** behaves similar to which **Haskell** feature?

	Type classes	1 (2.7 %)
➡	Lazy lists	30 (81.08 %)
	Pattern matching	0 (0 %)
	Monads	6 (16.22 %)

Average Gr
Standard Devia
Point Bis
Discrimination In

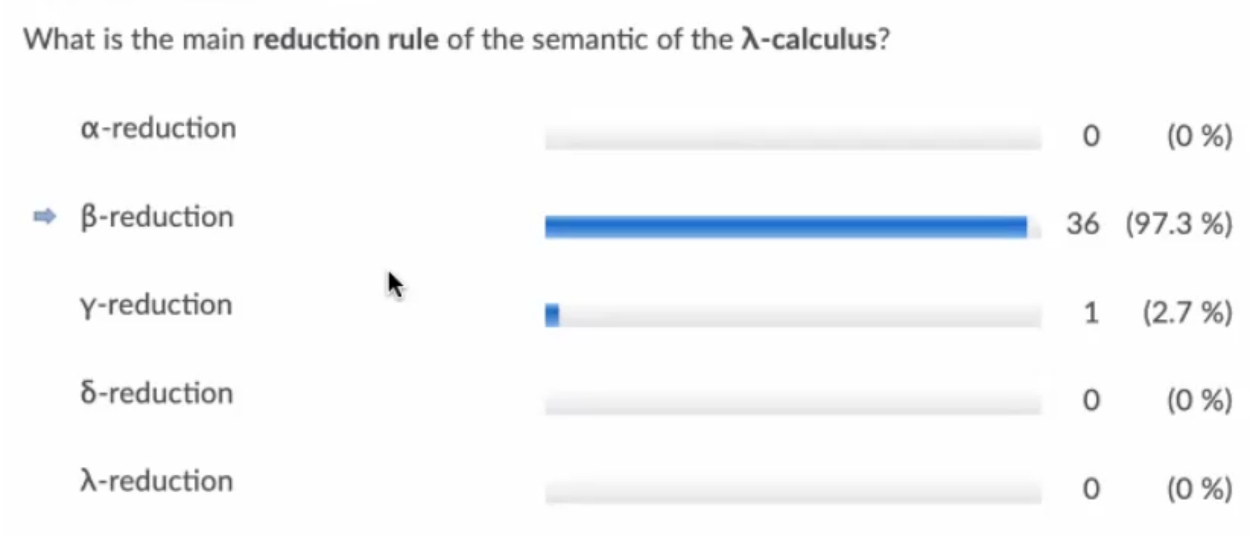
8. Genetic Functions in C++ are specifically using which keyword?



Generic functions in **C++** are specifically using which keyword?

➡	`template`	32 (86.49 %)
	`virtual`	1 (2.7 %)
	`generic`	4 (10.81 %)
	`const`	0 (0 %)

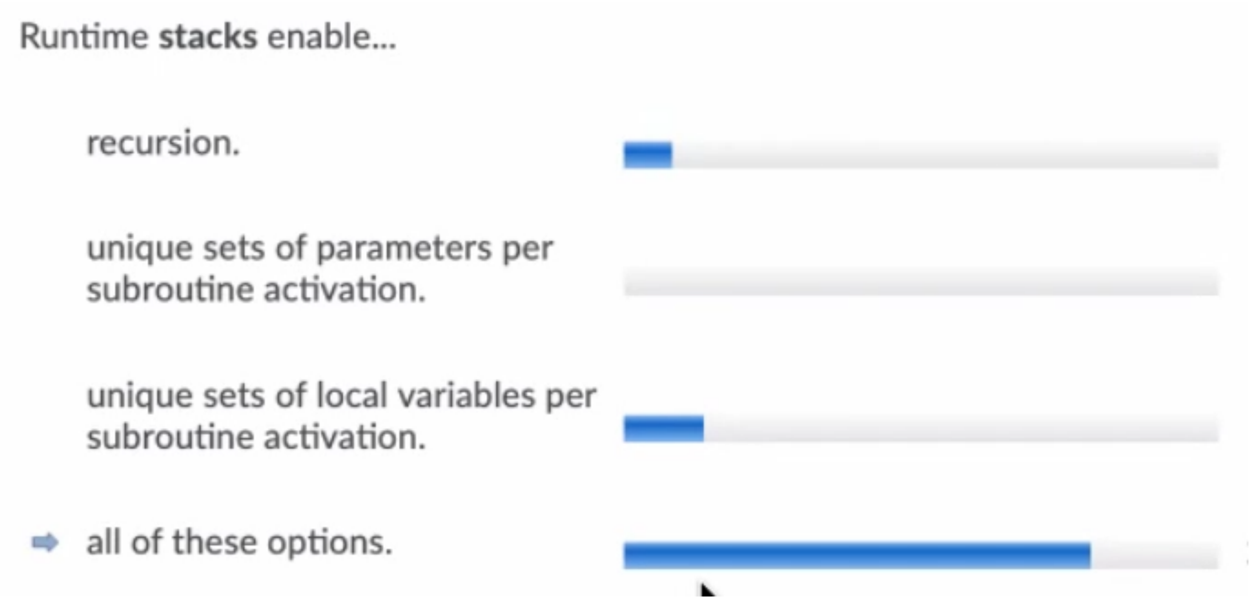
9. What is the main reduction rule of the semantic of the λ-calculus?



What is the main **reduction rule** of the semantic of the **λ-calculus**?

	α-reduction	0 (0 %)
➡	β-reduction	36 (97.3 %)
	γ-reduction	1 (2.7 %)
	δ-reduction	0 (0 %)
	λ-reduction	0 (0 %)

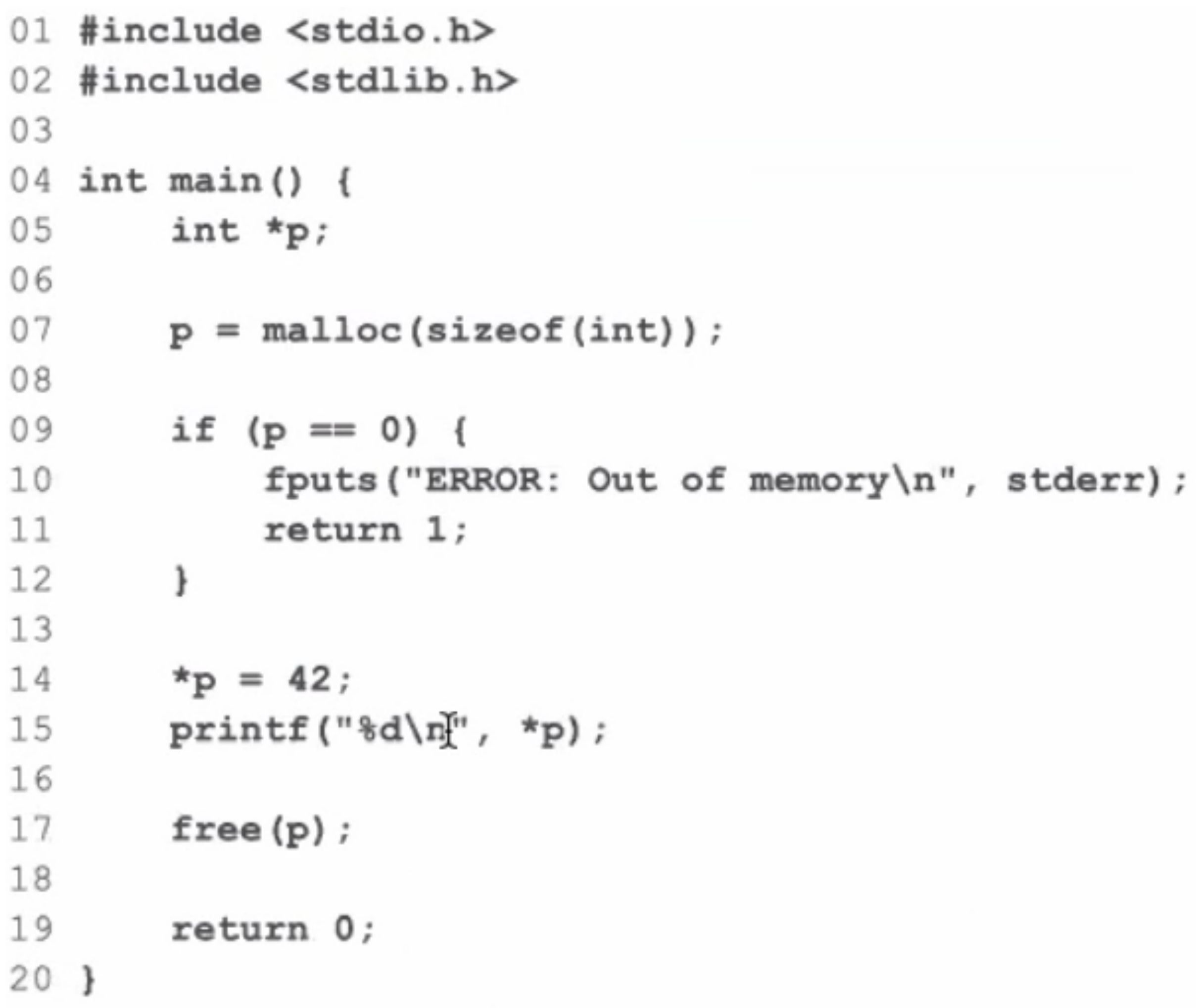
10. Runtime stacks enable...



Runtime **stacks** enable...

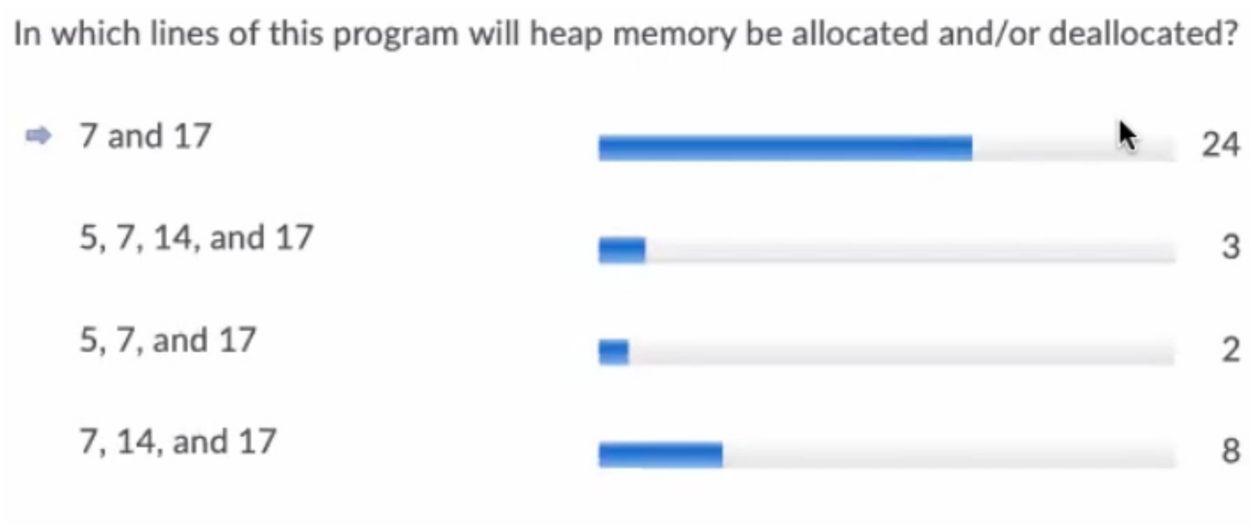
- recursion.
- unique sets of parameters per subroutine activation.
- unique sets of local variables per subroutine activation.
- [✓ all of these options.]

Consider the following C program:



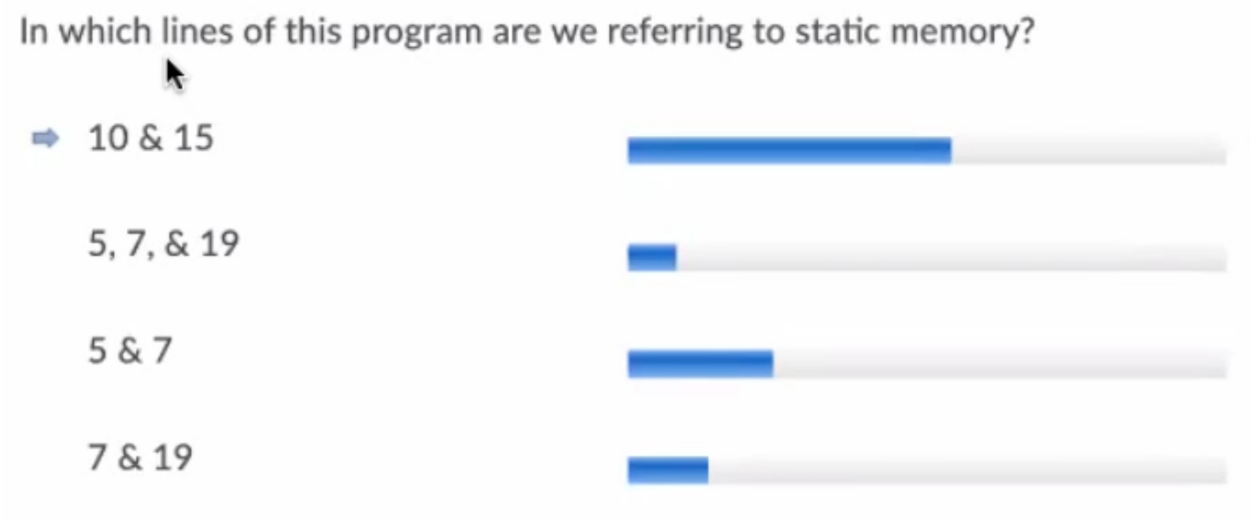
```
01 #include <stdio.h>
02 #include <stdlib.h>
03
04 int main() {
05     int *p;
06
07     p = malloc(sizeof(int));
08
09     if (p == 0) {
10         fputs("ERROR: Out of memory\n", stderr);
11         return 1;
12     }
13
14     *p = 42;
15     printf("%d\n", *p);
16
17     free(p);
18
19     return 0;
20 }
```

11. In which lines of this program will heap memory be allocated and/or deallocated?


In which lines of this program will heap memory be allocated and/or deallocated?

➡ 7 and 17	24
5, 7, 14, and 17	3
5, 7, and 17	2
7, 14, and 17	8

12. In which lines of this program are we referring to static memory?


In which lines of this program are we referring to static memory?

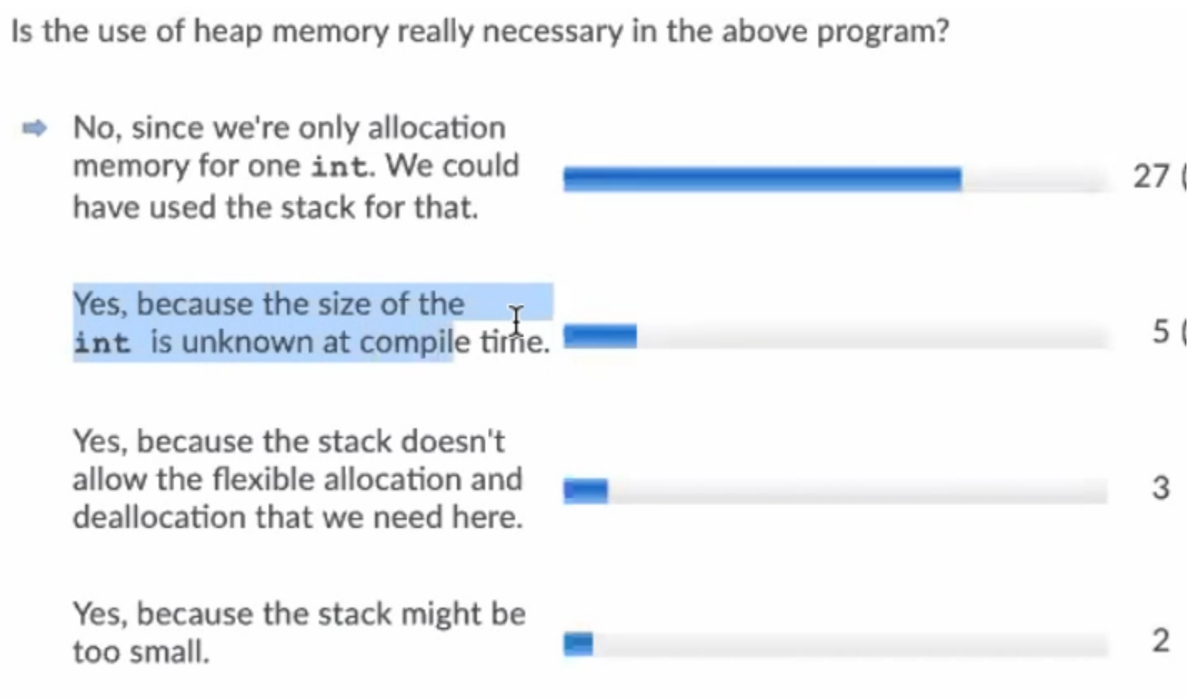
➡ 10 & 15

5, 7, & 19

5 & 7

7 & 19

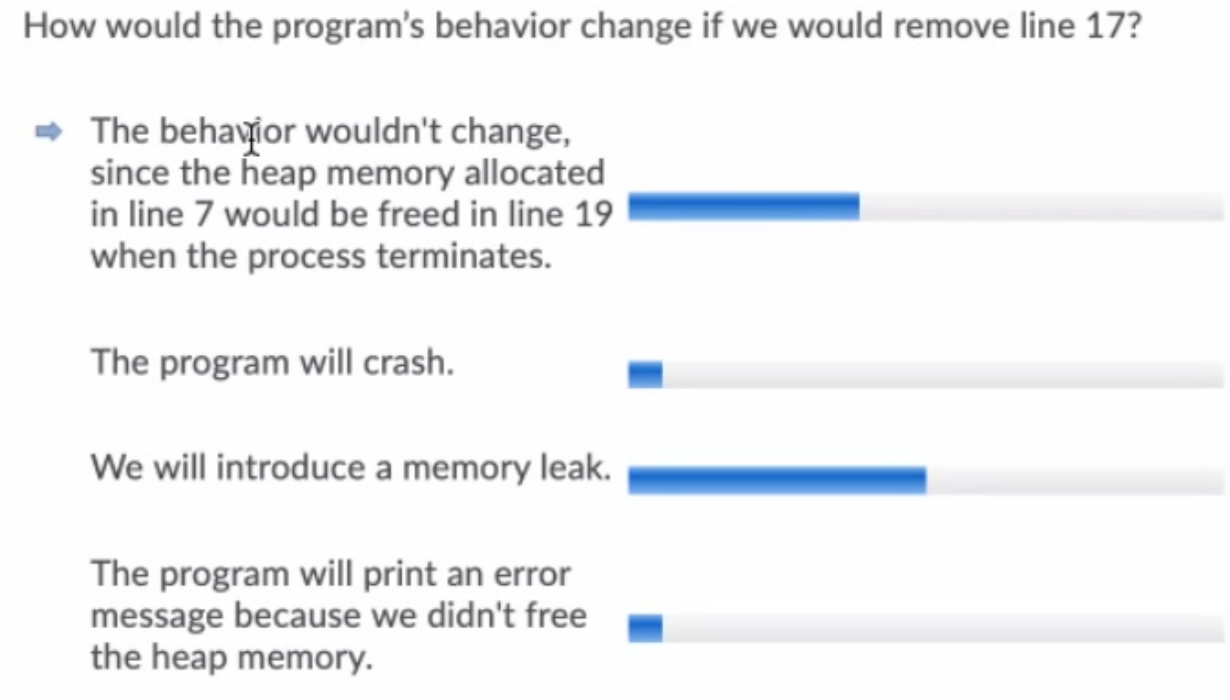
13. Is the use of heap memory really necessary in the above program?



Is the use of heap memory really necessary in the above program?

➡ No, since we're only allocation memory for one `int`. We could have used the stack for that.	27 (
Yes, because the size of the `int` is unknown at compile time.	5 (
Yes, because the stack doesn't allow the flexible allocation and deallocation that we need here.	3
Yes, because the stack might be too small.	2

14. How would the program's behavior change if we would remove line 17?



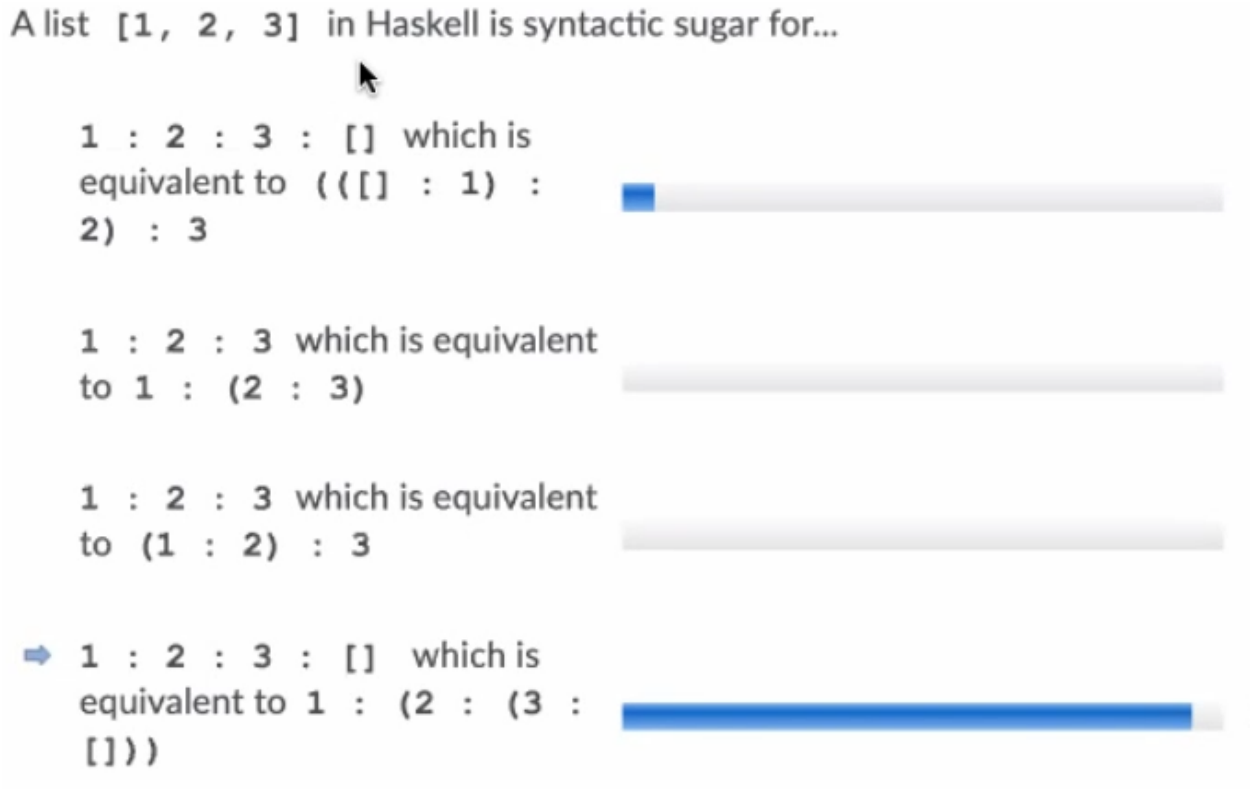
How would the program's behavior change if we would remove line 17?

- ➡ The behavior wouldn't change, since the heap memory allocated in line 7 would be freed in line 19 when the process terminates.
- The program will crash.
- We will introduce a memory leak.
- The program will print an error message because we didn't free the heap memory.

15. Match the programming language to the lambda expressions.
    a. [](auto x){ return x * x }
        i. C++
    b. x => x * x
        i. C#
    c. Lambda x: x * x

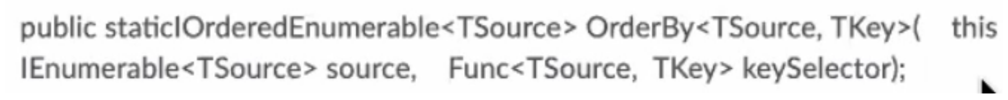
        i. Python
    d. \x -> x * x
        i. Haskell

16. A list [1, 2, 3] in Haskell is syntactic sugar for…



A list `[1, 2, 3]` in Haskell is syntactic sugar for...

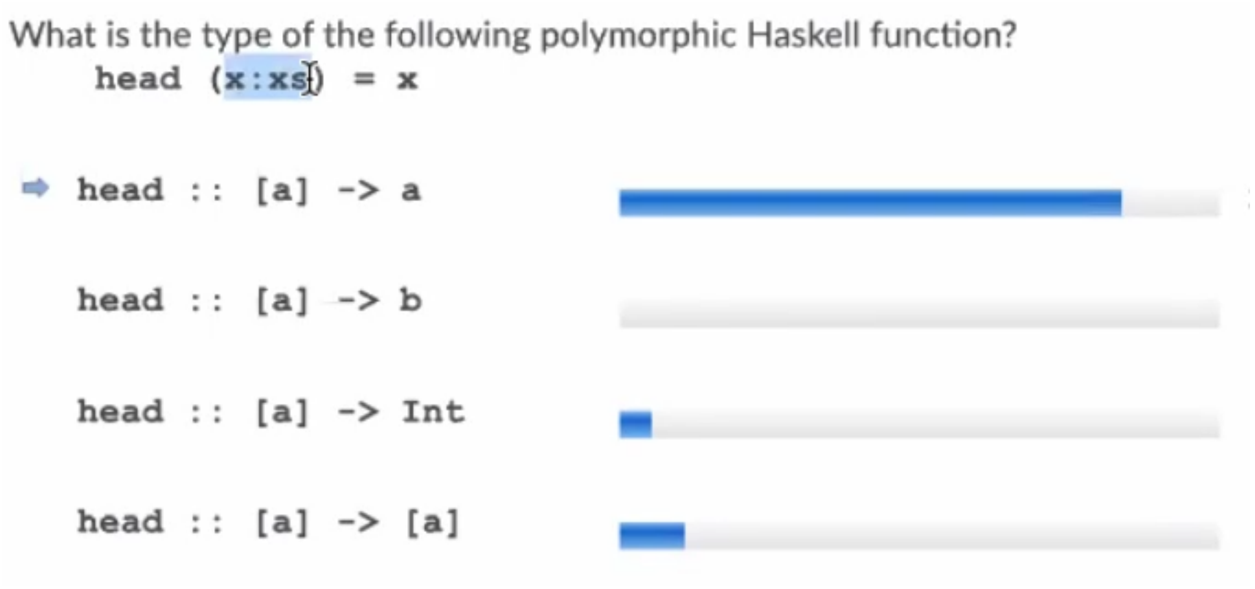
- `1 : 2 : 3 : []` which is equivalent to `(([] : 1) : 2) : 3`
- `1 : 2 : 3` which is equivalent to `1 : (2 : 3)`
- `1 : 2 : 3` which is equivalent to `(1 : 2) : 3`
- [✓ `1 : 2 : 3 : []` which is equivalent to `1 : (2 : (3 : []))`]

17. Match the following sort functions with the programming languages they're written in.
    a. void qsort(void *a, size_t n, size_t width, int (*comp)(const void *, const void *));
        i. C
    b. sorted(iterable[, key][, reverse])
        i. Python
    c. template<class RandomAccessIterator, class Compare> void sort (RandomAccessiterator first, Random AccessIterator last, Compare comp);
        i. C++
    d. sort:: Ord a => [a] -> [a]
        i. Haskell
    e. 

public staticIOrderedEnumerable<TSource> OrderBy<TSource, TKey>(this IEnumerable<TSource> source, Func<TSource, TKey> keySelector);

        i. C#

18. What is the type of the following polymorphic Haskell function?

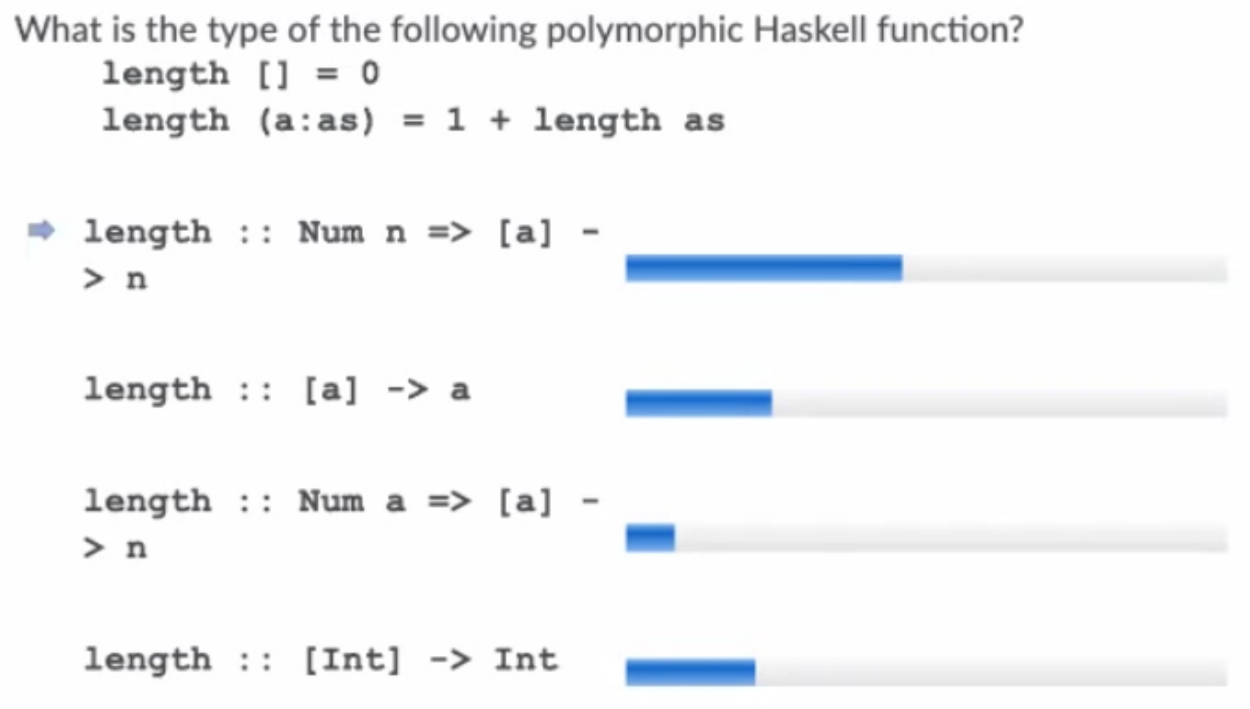


What is the type of the following polymorphic Haskell function?

```
head (x:xs) = x
```

- [✓ `head :: [a] -> a`]
- `head :: [a] -> b`
- `head :: [a] -> Int`
- `head :: [a] -> [a]`

19. What is the type of the following polymorphic Haskell function?

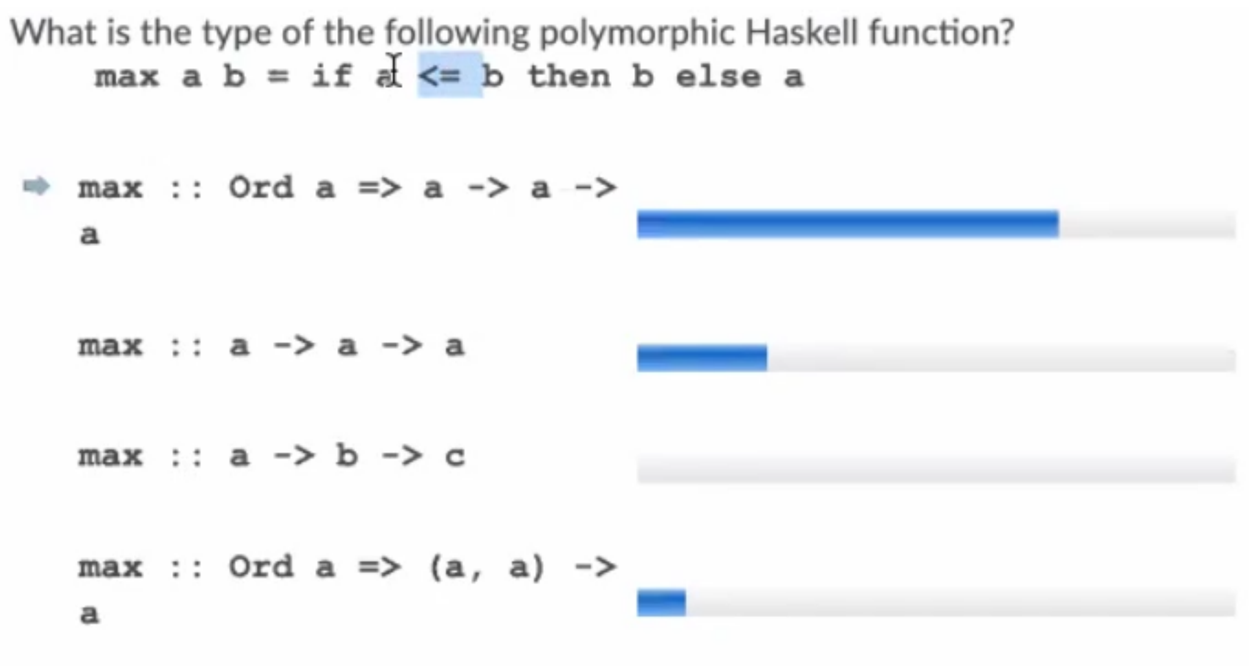


What is the type of the following polymorphic Haskell function?

```
length [] = 0
length (a:as) = 1 + length as
```

- [✓ `length :: Num n => [a] -> n`]
- `length :: [a] -> a`
- `length :: Num a => [a] -> n`
- `length :: [Int] -> Int`

20. What is the type of the following polymorphic Haskell function?



What is the type of the following polymorphic Haskell function?

```
max a b = if a <= b then b else a
```

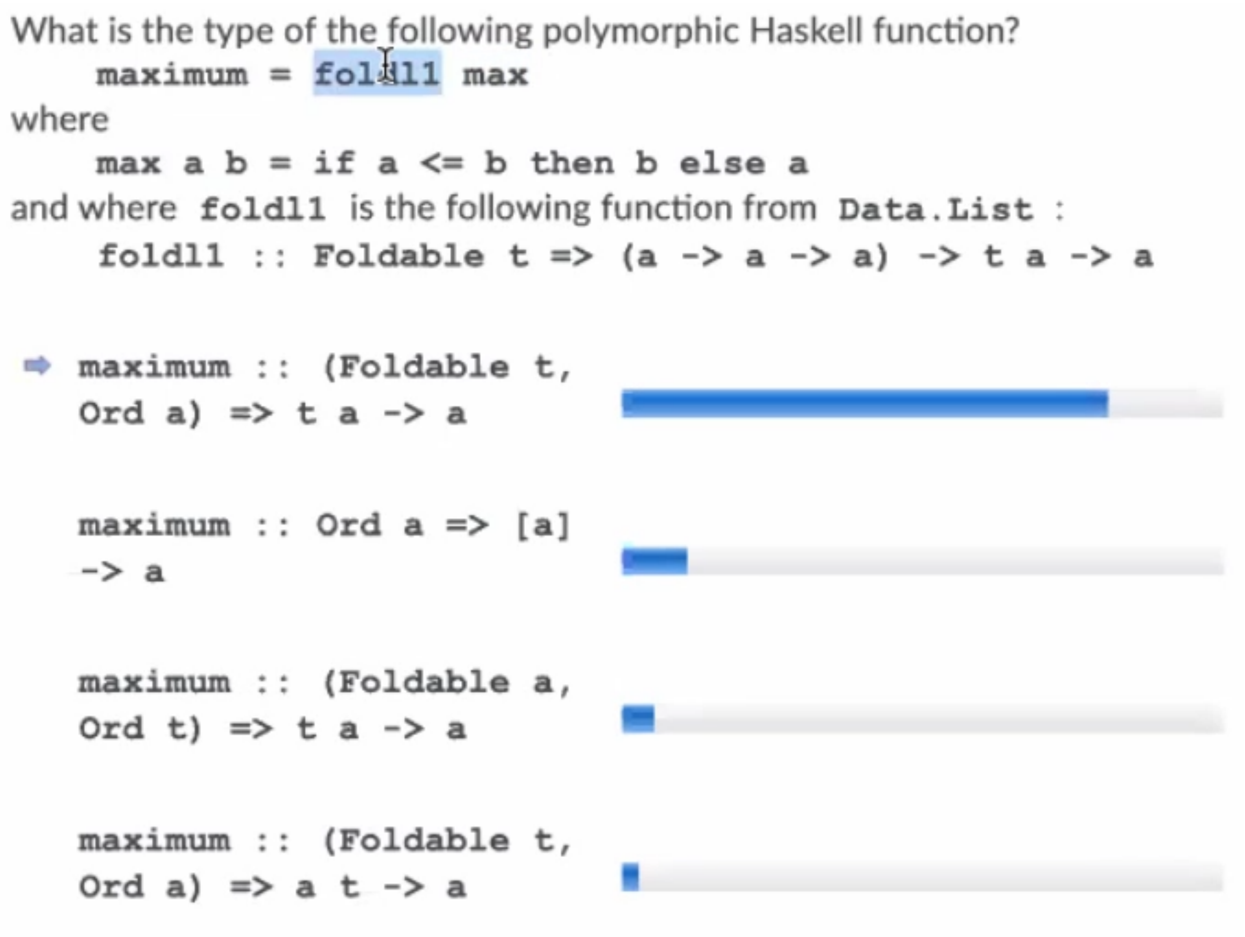
[✓ `max :: Ord a => a -> a -> a`]

`max :: a -> a -> a`

`max :: a -> b -> c`

`max :: Ord a => (a, a) -> a`

21. What is the type of the following polymorphic Haskell function?



What is the type of the following polymorphic Haskell function?

```
maximum = foldl1 max
```

where

```
max a b = if a <= b then b else a
```

and where `foldl1` is the following function from `Data.List` :

```
foldl1 :: Foldable t => (a -> a -> a) -> t a -> a
```

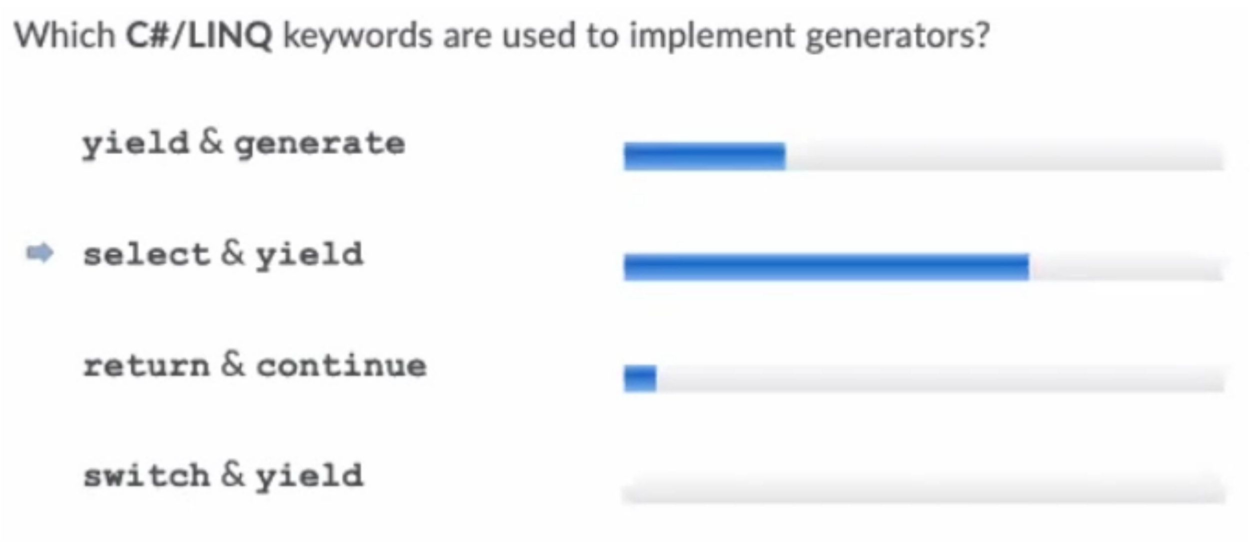
[✓ `maximum :: (Foldable t, Ord a) => t a -> a`]

`maximum :: Ord a => [a] -> a`

`maximum :: (Foldable a, Ord t) => t a -> a`

`maximum :: (Foldable t, Ord a) => a t -> a`

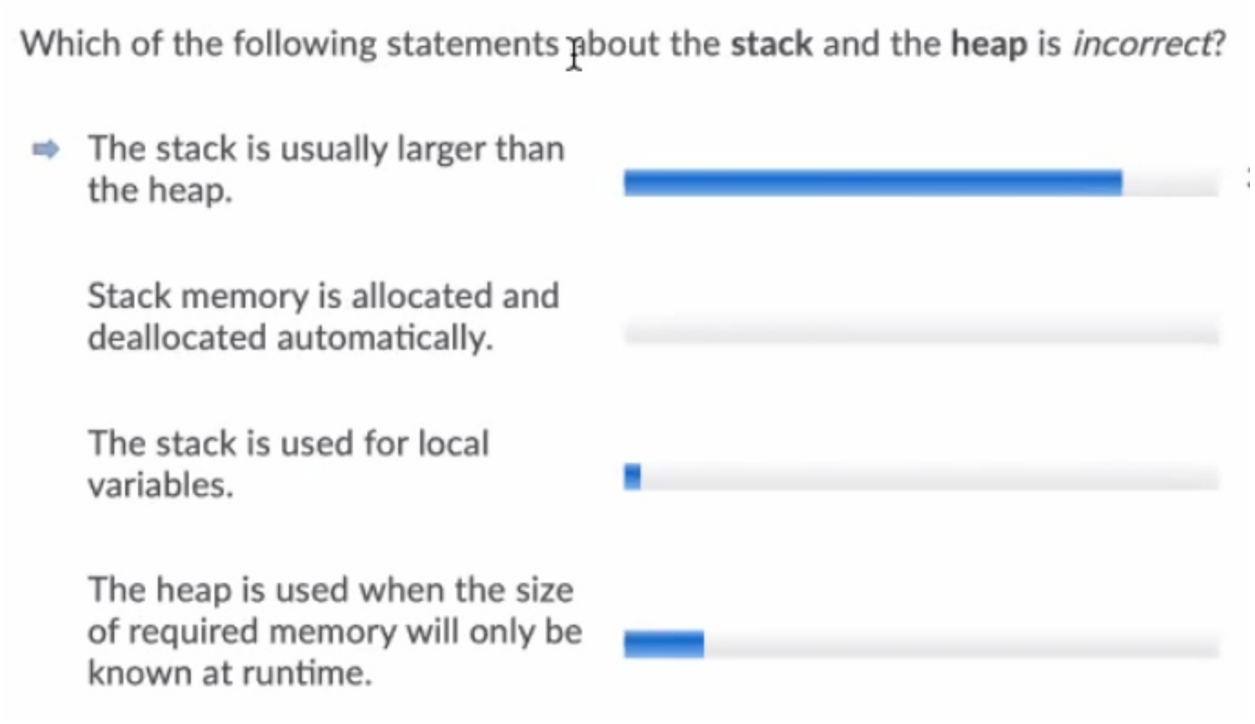
22. Which C#/LINQ keywords are used to implement generators?



Which **C#/LINQ** keywords are used to implement generators?

- `yield` & `generate`
- ➡ `select` & `yield`
- `return` & `continue`
- `switch` & `yield`

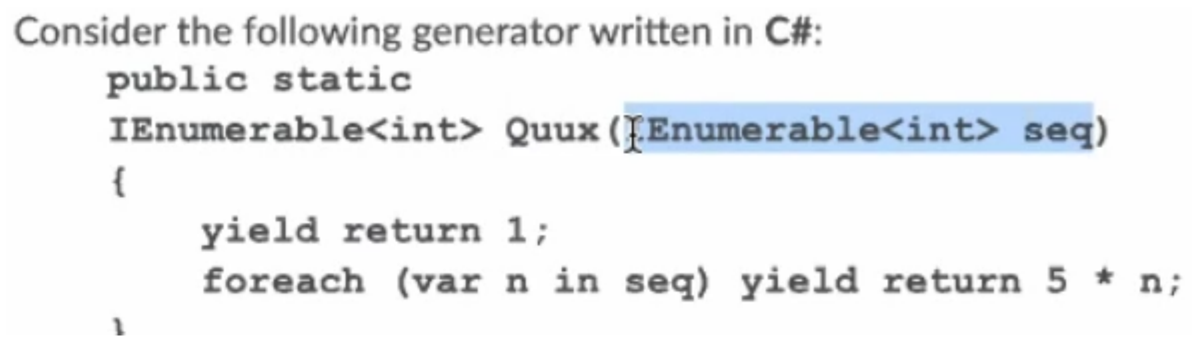
23. Which of the following statements about the stack and the heap is incorrect?



Which of the following statements about the **stack** and the **heap** is *incorrect*?

- ➡ The stack is usually larger than the heap.
- Stack memory is allocated and deallocated automatically.
- The stack is used for local variables.
- The heap is used when the size of required memory will only be known at runtime.

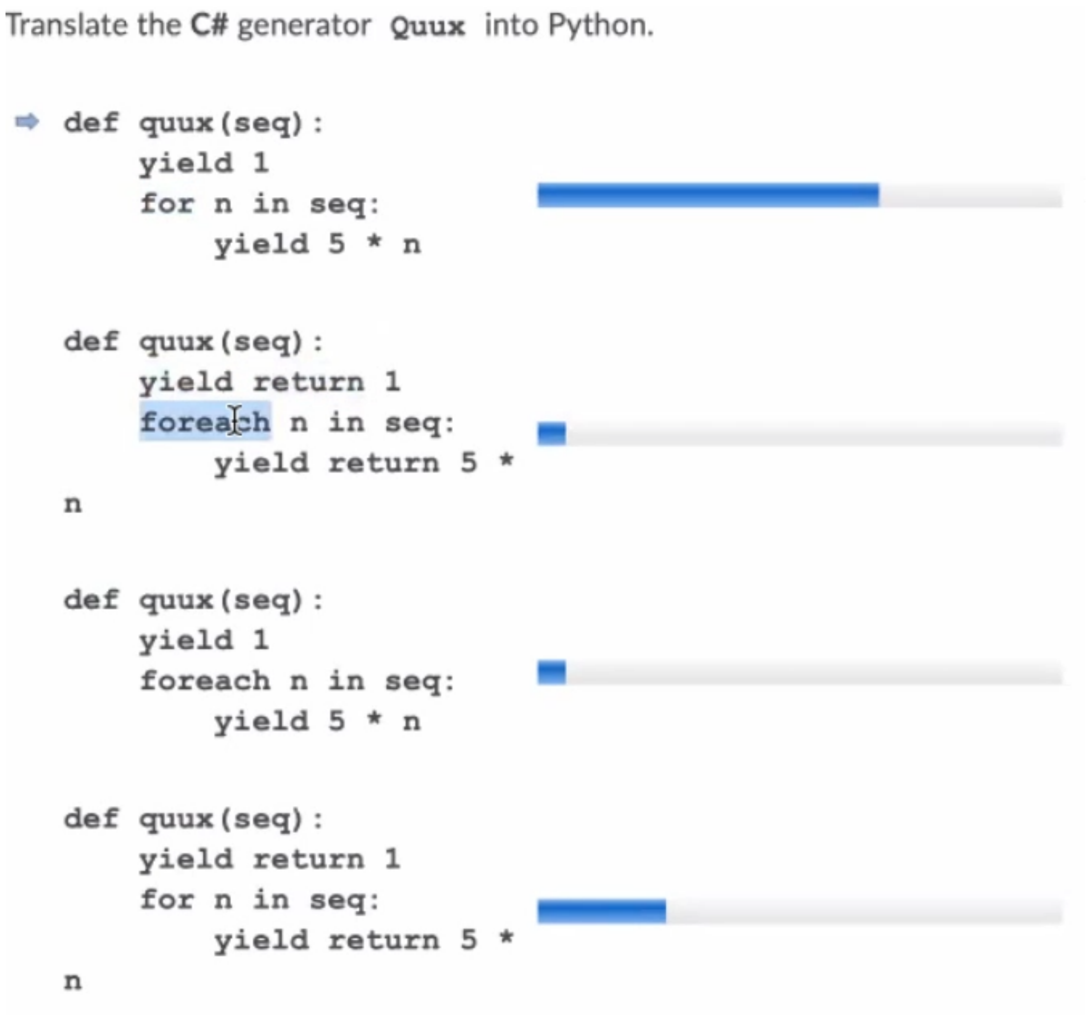
Consider the following generator written in C#:



Consider the following generator written in **C#**:

```
public static
IEnumerable<int> Quux(IEnumerable<int> seq)
{
    yield return 1;
    foreach (var n in seq) yield return 5 * n;
}
```

24. Translate the C# generator Quux into Python.



Translate the **C#** generator `Quux` into Python.

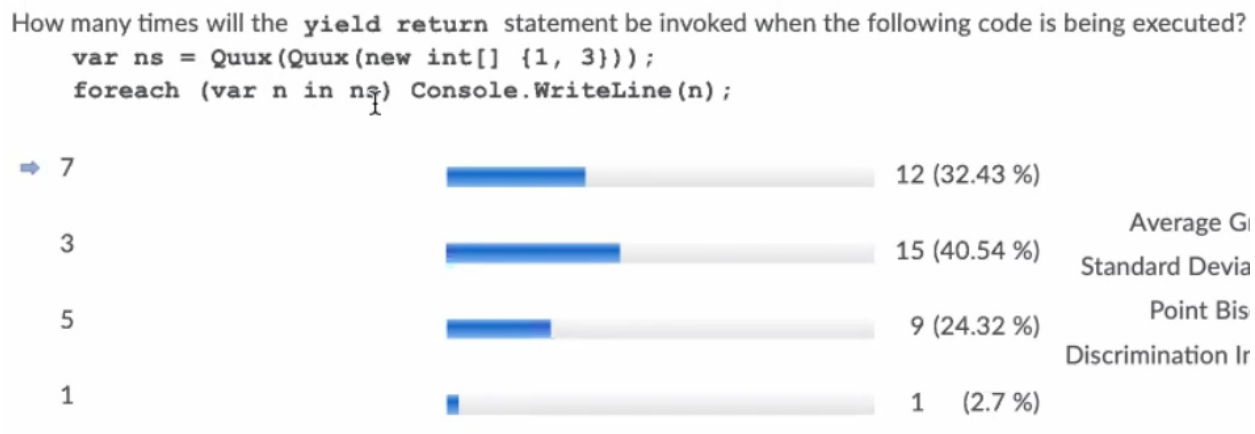
```
➡ def quux(seq):
      yield 1
      for n in seq:
          yield 5 * n

  def quux(seq):
      yield return 1
      foreach n in seq:
          yield return 5 *
  n

  def quux(seq):
      yield 1
      foreach n in seq:
          yield 5 * n

  def quux(seq):
      yield return 1
      for n in seq:
          yield return 5 *
  n
```

25. How many times will the yield return statement be invoked when the following code is being executed
?



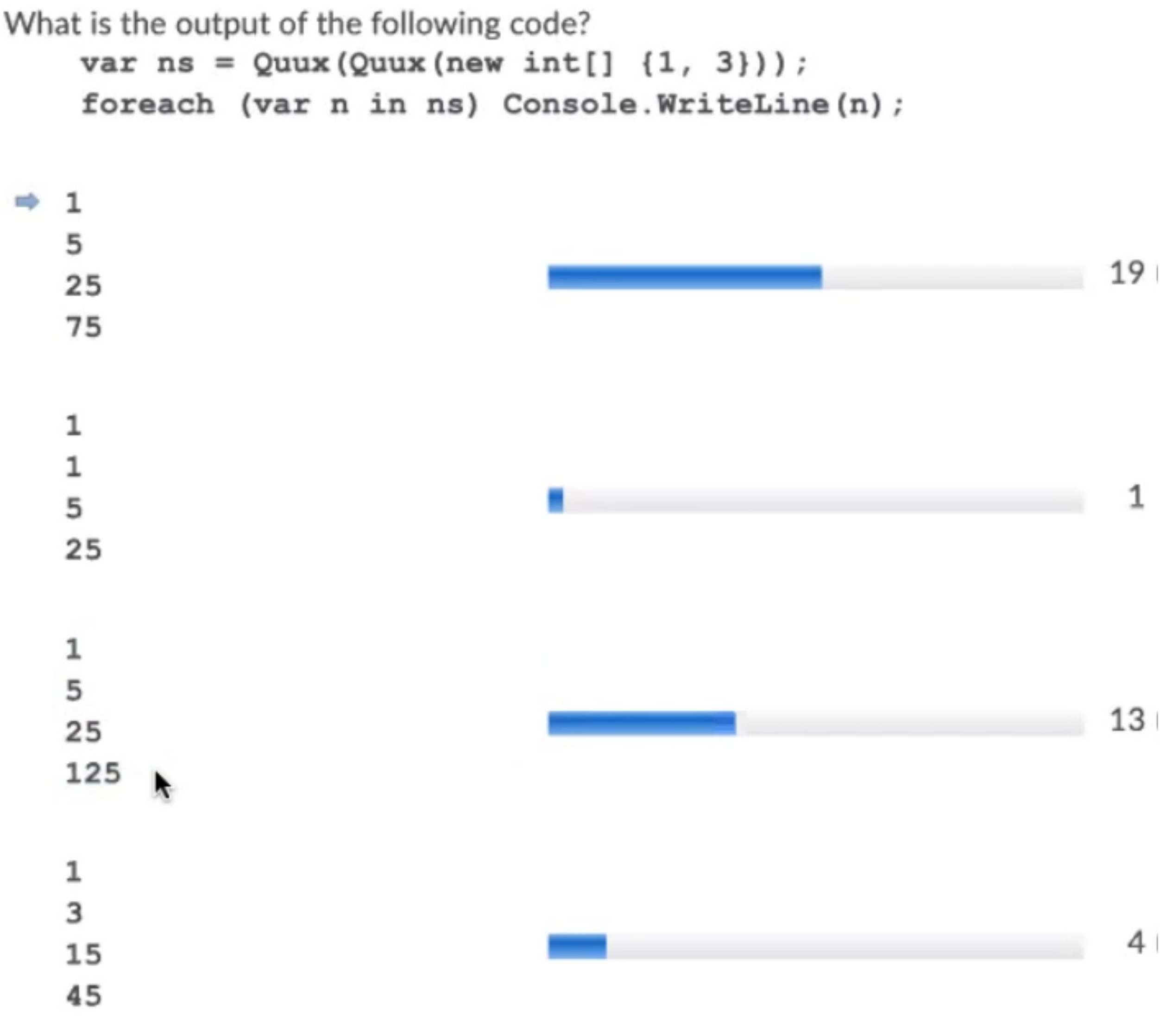
How many times will the `yield return` statement be invoked when the following code is being executed?

```
var ns = Quux(Quux(new int[] {1, 3}));
foreach (var n in ns) Console.WriteLine(n);
```

➡ 7	12 (32.43 %)
3	15 (40.54 %)
5	9 (24.32 %)
1	1 (2.7 %)

Average Gı
Standard Devia
Point Bis
Discrimination Ir

26. What is the output of the following code?



What is the output of the following code?

```
    var ns = Quux(Quux(new int[] {1, 3}));
    foreach (var n in ns) Console.WriteLine(n);
```

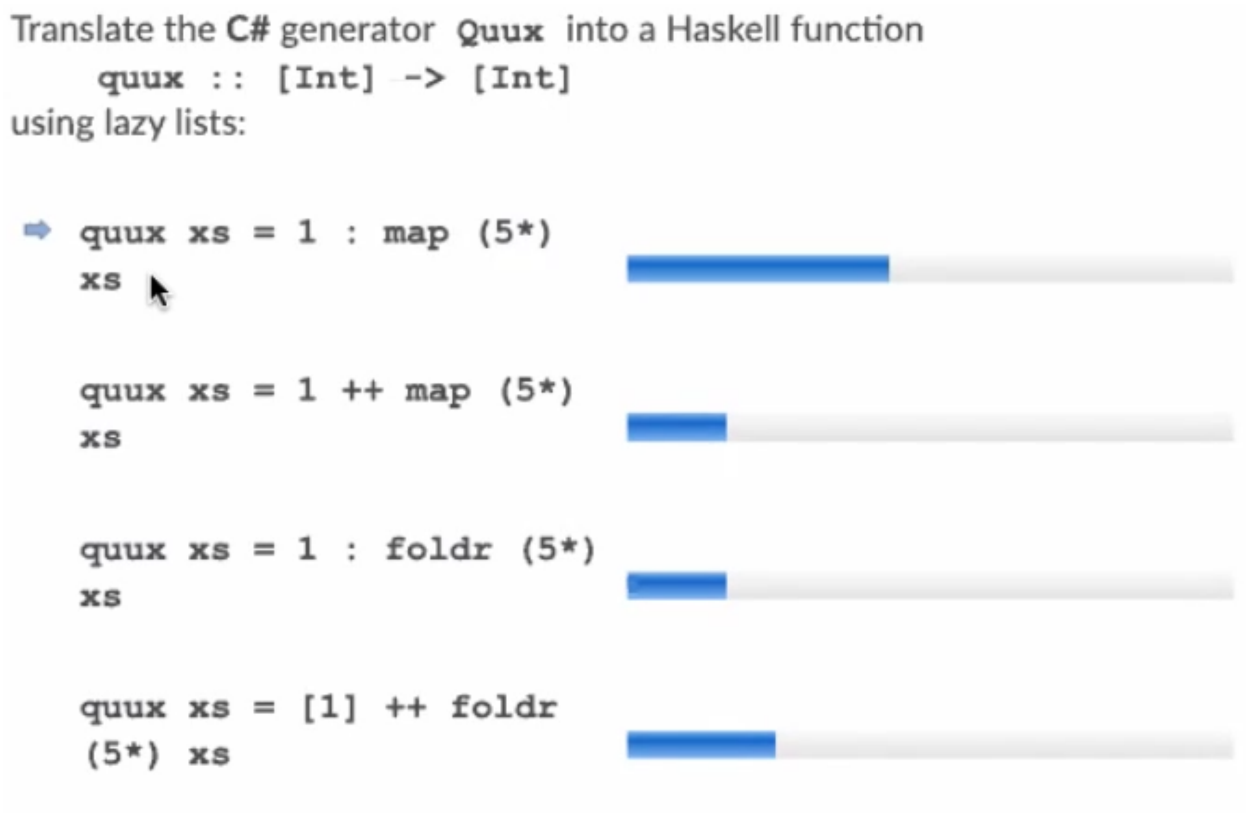
[✓
```
1
5
25
75
```
] 19

```
1
1
5
25
```
1

```
1
5
25
125
```
13

```
1
3
15
45
```
4

27. Translate the C# generator Quux into Haskell function



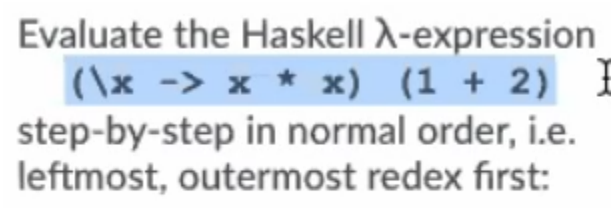
Translate the **C#** generator `Quux` into a Haskell function

```
    quux :: [Int] -> [Int]
```

using lazy lists:

- ➡ `quux xs = 1 : map (5*) xs`
- `quux xs = 1 ++ map (5*) xs`
- `quux xs = 1 : foldr (5*) xs`
- `quux xs = [1] ++ foldr (5*) xs`

Evaluate the Haskell λ-expression

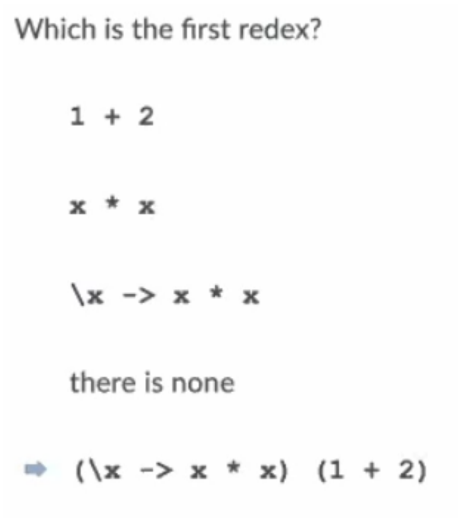


Evaluate the Haskell λ-expression

```
  (\x -> x * x) (1 + 2)
```

step-by-step in normal order, i.e. leftmost, outermost redex first:

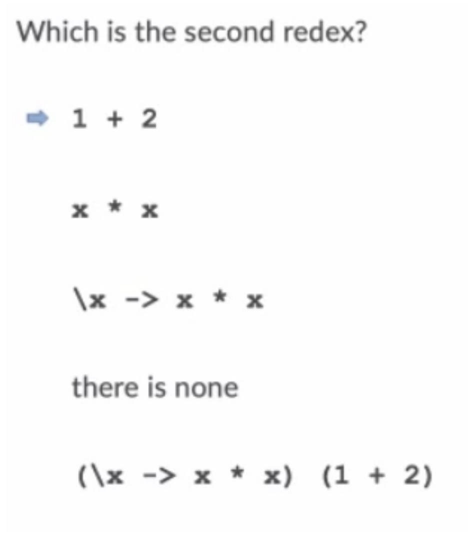
28. Which is the first redex?



Which is the first redex?

- `1 + 2`
- `x * x`
- `\x -> x * x`
- there is none
- ➡ `(\x -> x * x) (1 + 2)`

29. Which is the second redex?


Which is the second redex?

[✓ `1 + 2`]

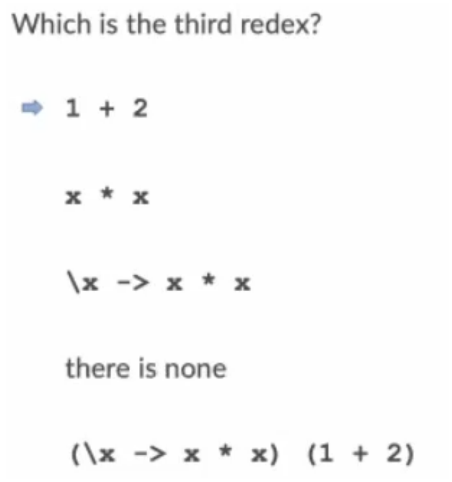
`x * x`

`\x -> x * x`

there is none

`(\x -> x * x) (1 + 2)`

30. Which is the third redex?


Which is the third redex?

[✓ `1 + 2`]

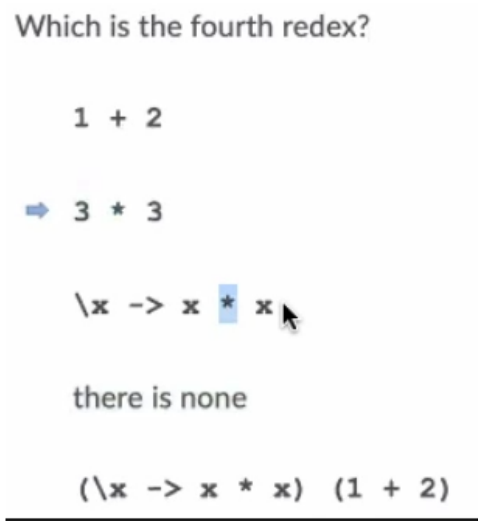
`x * x`

`\x -> x * x`

there is none

`(\x -> x * x) (1 + 2)`

31. Which is the fourth redex?


Which is the fourth redex?

`1 + 2`

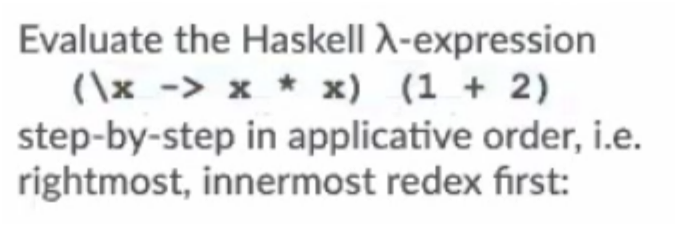
[✓ `3 * 3`]

`\x -> x * x`

there is none

`(\x -> x * x) (1 + 2)`

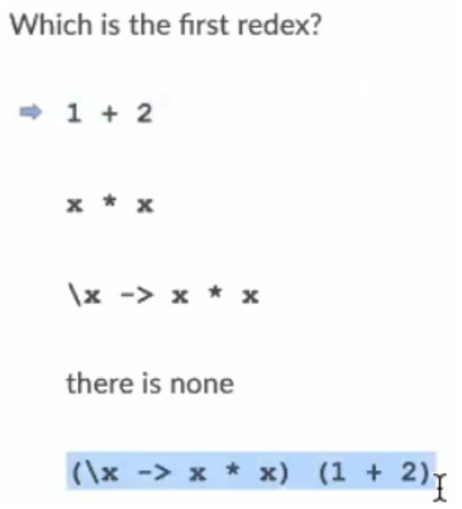
Evaluate the Haskell λ-expression


Evaluate the Haskell λ-expression

```
(\x -> x * x) (1 + 2)
```

step-by-step in applicative order, i.e. rightmost, innermost redex first:

32. Which is the fist redex?


Which is the first redex?

⇒ `1 + 2`

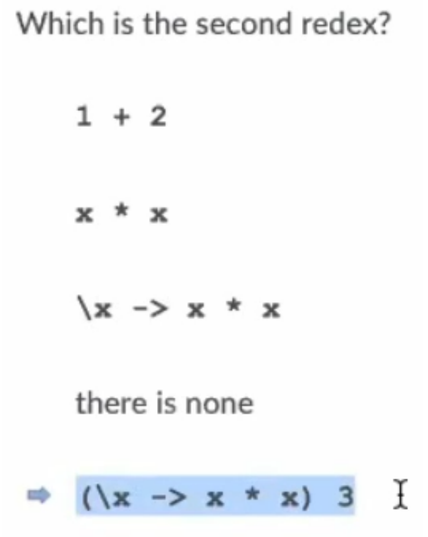
`x * x`

`\x -> x * x`

there is none

`(\x -> x * x) (1 + 2)`

33. Which is the second redex?


Which is the second redex?

`1 + 2`

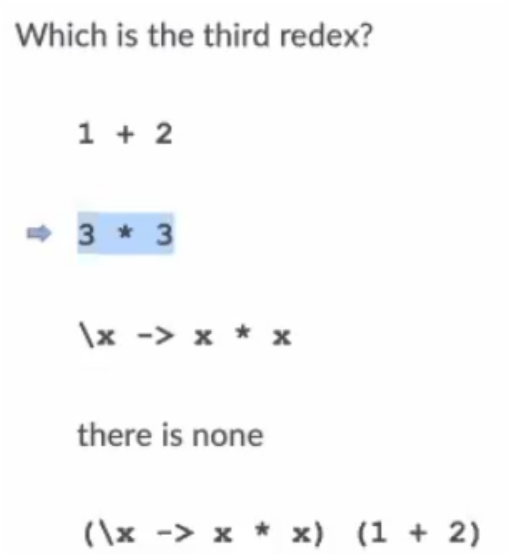
`x * x`

`\x -> x * x`

there is none

⇒ `(\x -> x * x) 3`

34. Which is the thirth redex?


Which is the third redex?

`1 + 2`

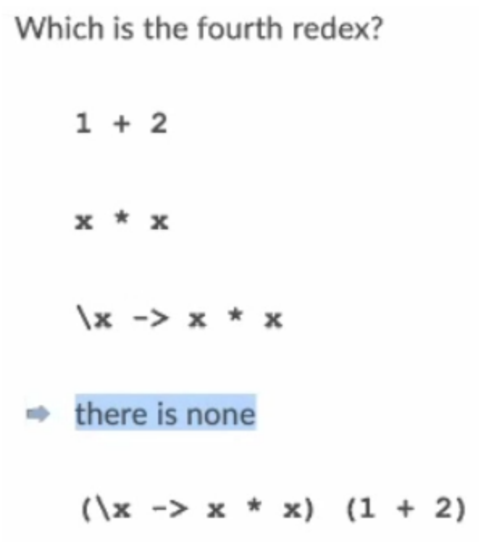
[✓ `3 * 3`]

`\x -> x * x`

there is none

`(\x -> x * x) (1 + 2)`

35. Which is the fourth redex?


Which is the fourth redex?

`1 + 2`

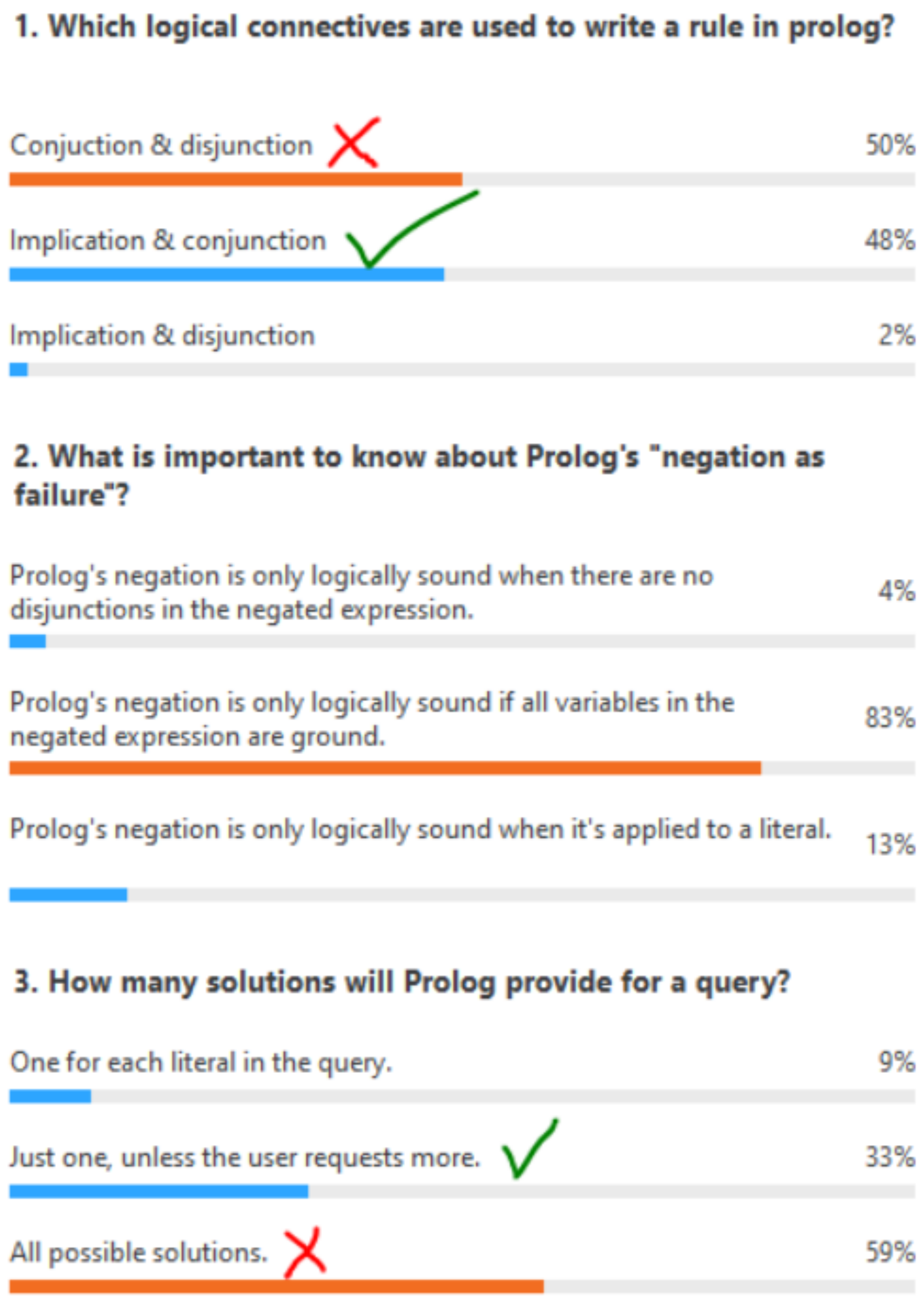
`x * x`

`\x -> x * x`

[✓ there is none]

`(\x -> x * x) (1 + 2)`

EXTRAS
Which logical connectives are used to write a rule in prolog?
What is important to know about Prolog’s “negation as failure”?
How many solutions will Prolog provide for a query?



1. Which logical connectives are used to write a rule in prolog?

Conjuction & disjunction ✗	50%
[✓ Implication & conjunction]	48%
Implication & disjunction	2%

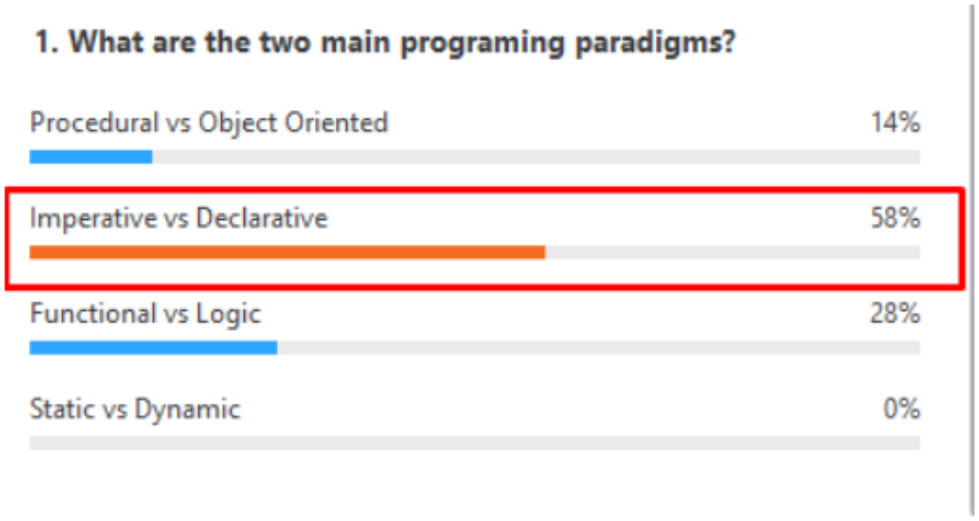
2. What is important to know about Prolog's "negation as failure"?

Prolog's negation is only logically sound when there are no disjunctions in the negated expression.	4%
Prolog's negation is only logically sound if all variables in the negated expression are ground.	83%
Prolog's negation is only logically sound when it's applied to a literal.	13%

3. How many solutions will Prolog provide for a query?

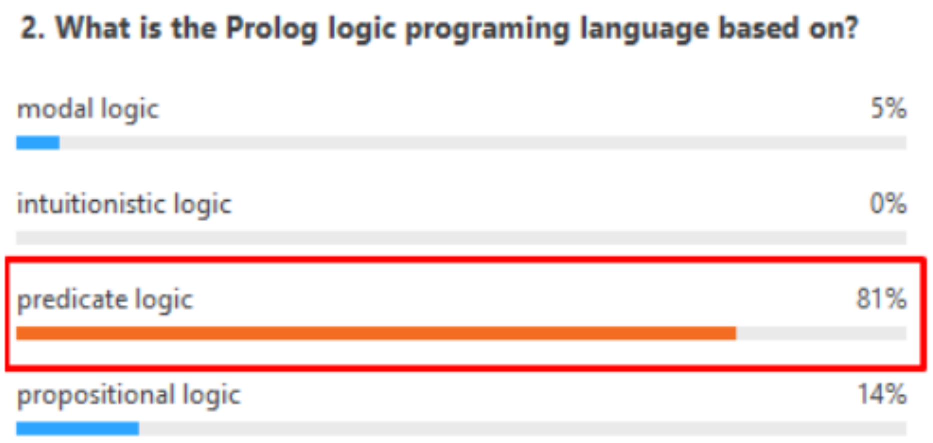
One for each literal in the query.	9%
[✓ Just one, unless the user requests more.]	33%
All possible solutions. ✗	59%

What are the two main programing paradigms?
What is the Prolog logic programing language based on?



1. What are the two main programing paradigms?

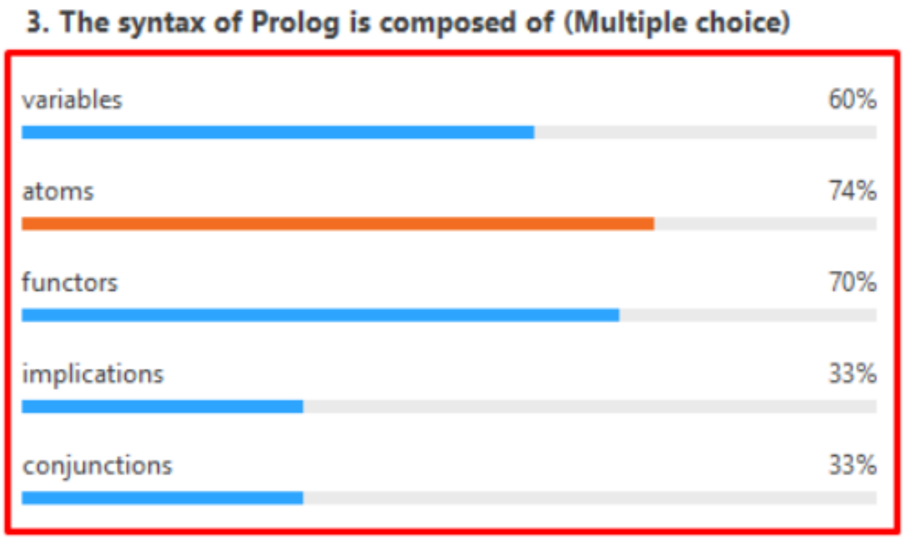
Procedural vs Object Oriented	14%
[✓ Imperative vs Declarative]	58%
Functional vs Logic	28%
Static vs Dynamic	0%



2. What is the Prolog logic programing language based on?

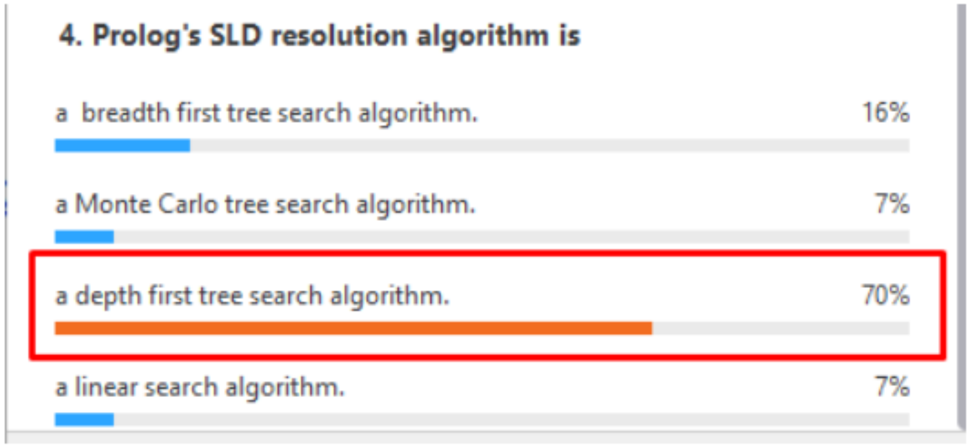
modal logic	5%
intuitionistic logic	0%
[✓ predicate logic]	81%
propositional logic	14%

The Syntax of prolog is composed of (Multiple choice)
Prolog’s SLD resolution algorithm is



3. The syntax of Prolog is composed of (Multiple choice)

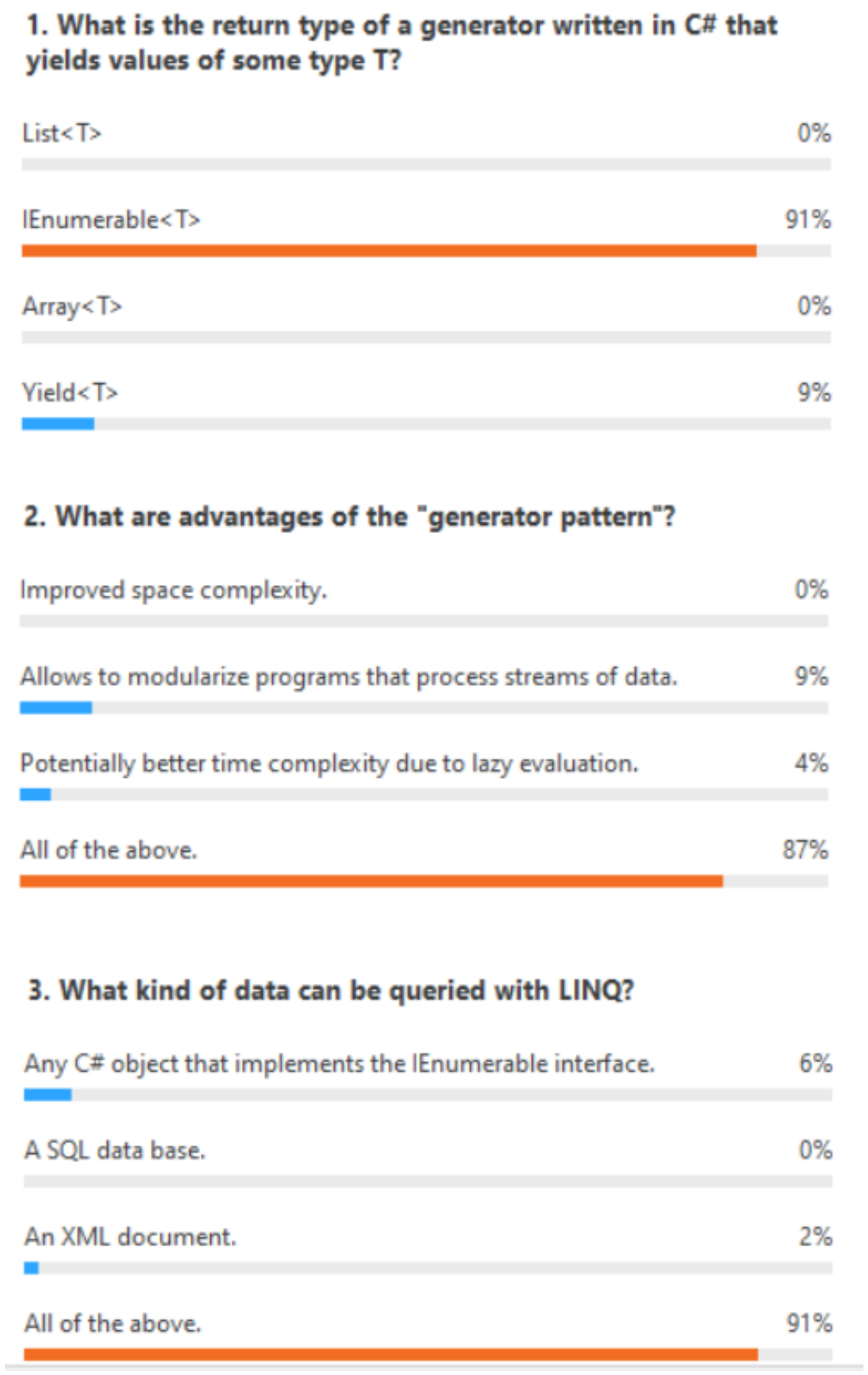
variables	60%
atoms	74%
functors	70%
implications	33%
conjunctions	33%



4. Prolog's SLD resolution algorithm is

a breadth first tree search algorithm.	16%
a Monte Carlo tree search algorithm.	7%
a depth first tree search algorithm.	70%
a linear search algorithm.	7%

What is the return type of a generator written in C# that yields values of some type T?
What are advantages of the “generator pattern”?
What kind of data can be queried with LINQ?



1. What is the return type of a generator written in C# that yields values of some type T?

List<T>	0%
IEnumerable<T>	91%
Array<T>	0%
Yield<T>	9%

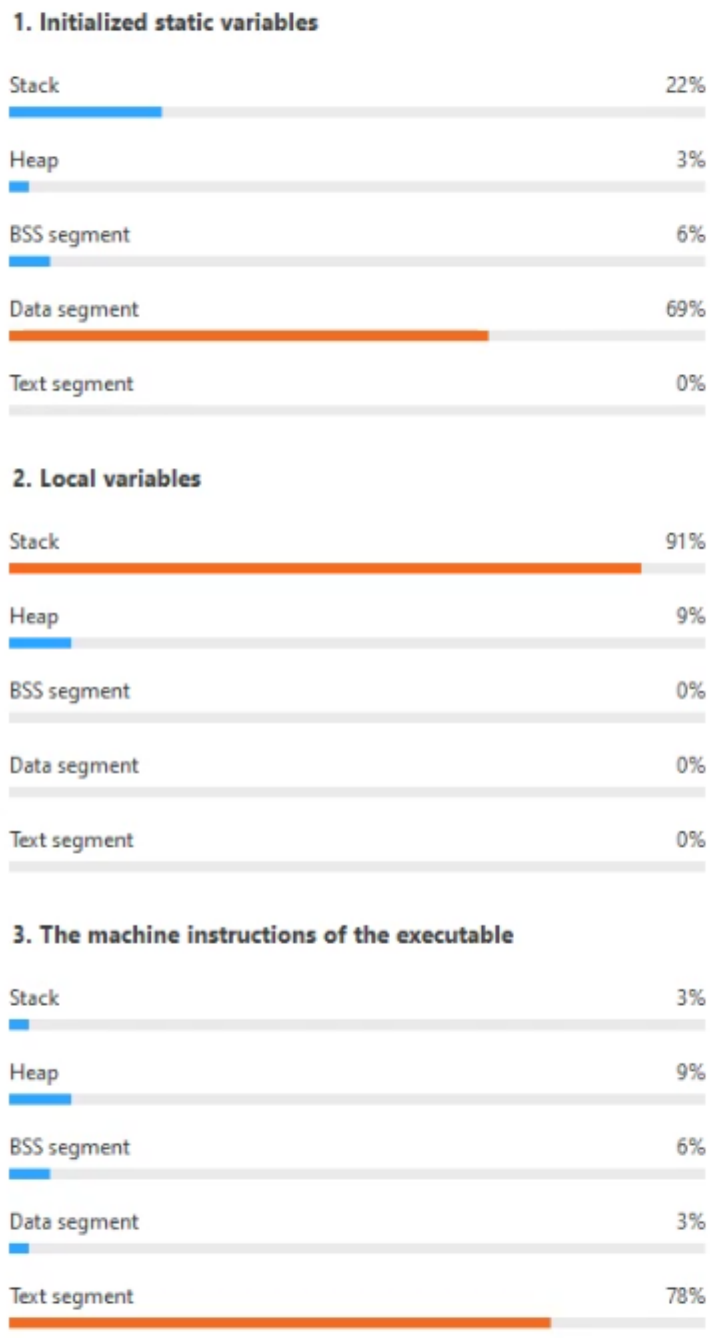
2. What are advantages of the "generator pattern"?

Improved space complexity.	0%
Allows to modularize programs that process streams of data.	9%
Potentially better time complexity due to lazy evaluation.	4%
All of the above.	87%

3. What kind of data can be queried with LINQ?

Any C# object that implements the IEnumerable interface.	6%
A SQL data base.	0%
An XML document.	2%
All of the above.	91%

Initialized static variables
Local Variables
The machine instructions of the executable
Uninitialized static variables
BSS segment


1. Initialized static variables

Stack	22%
Heap	3%
BSS segment	6%
Data segment	69%
Text segment	0%

2. Local variables

Stack	91%
Heap	9%
BSS segment	0%
Data segment	0%
Text segment	0%

3. The machine instructions of the executable

Stack	3%
Heap	9%
BSS segment	6%
Data segment	3%
Text segment	78%