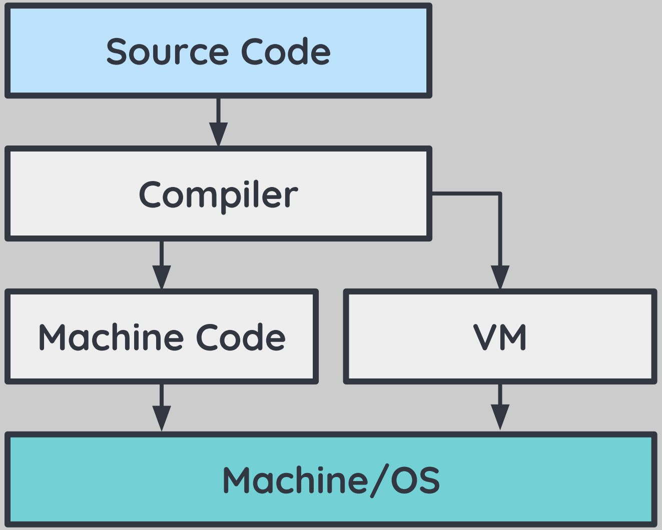
A META-INTERPRETER

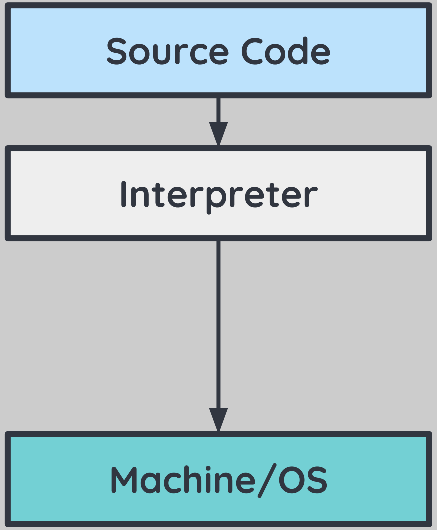
Programming II - Spring 2018

λ ID1019 KTH

How do you run?



Compiled language



Interpreted language

Compile or interpret?

C	**COMPILED**
Java	**COMPILED (VM)**
C++	**COMPILED**
Pascal	**COMPILED**
Ruby	**INTERPRETED**
Elixir/Erlang	**COMPILED (VM)**
Python	**INTERPRETED**
JavaScript	**INTERPRETED**
Go	**COMPILED**
PHP	**INTERPRETED**
Rust	**COMPILED**

A basic language

Sequence

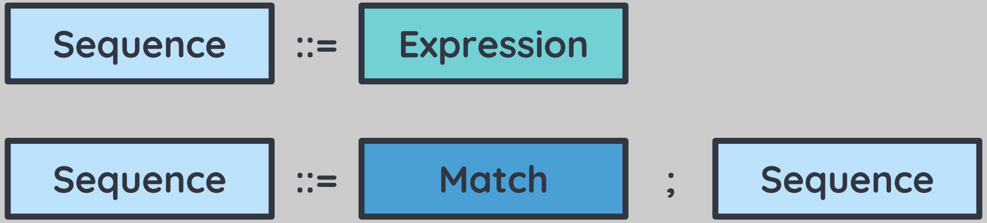
```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```

```
1.  x = foo;
2.  y = :nil;
3.  {z, _} = {:bar, :grk};
4.  {x, {z, y}}
```

Pattern matching expressions

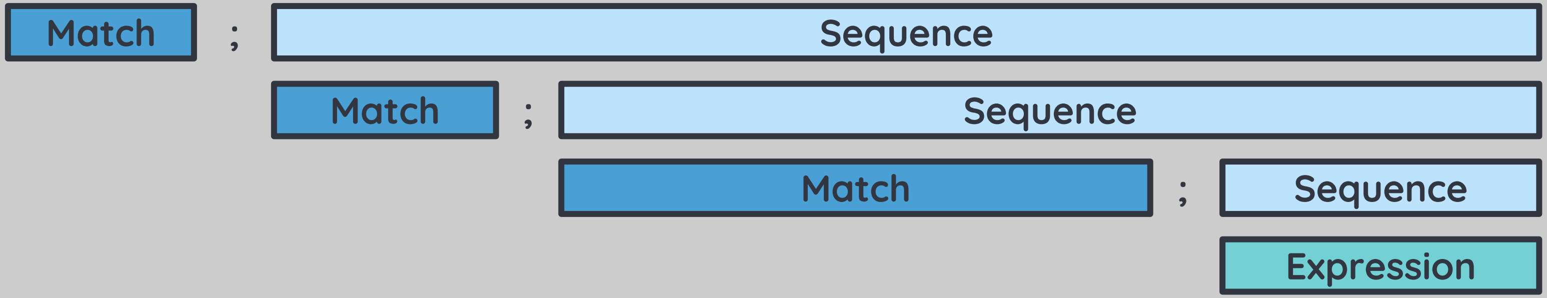
Single expression

A sequence



Example

```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```



A match

Match ::= Pattern = Expression

Example

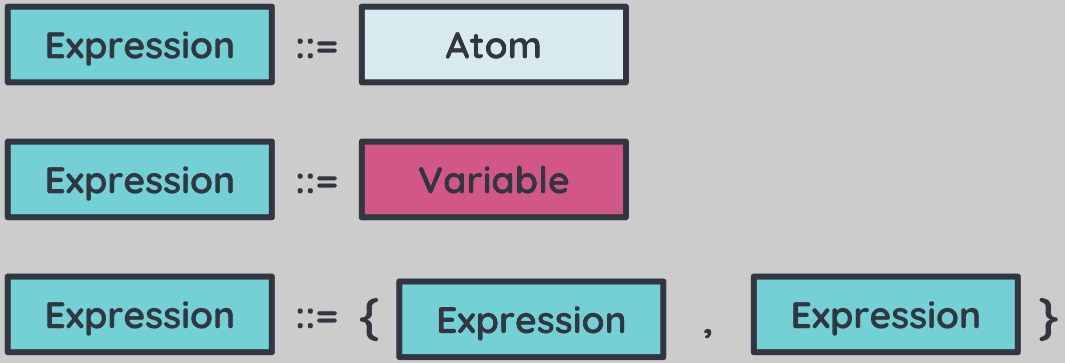
```
x = foo
```

Pattern = Expression

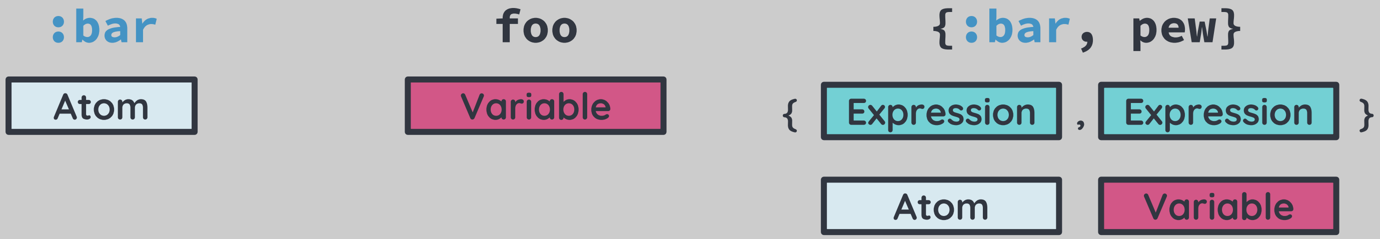
```
{z, _} = {:bar, :grk}
```

Pattern = Expression

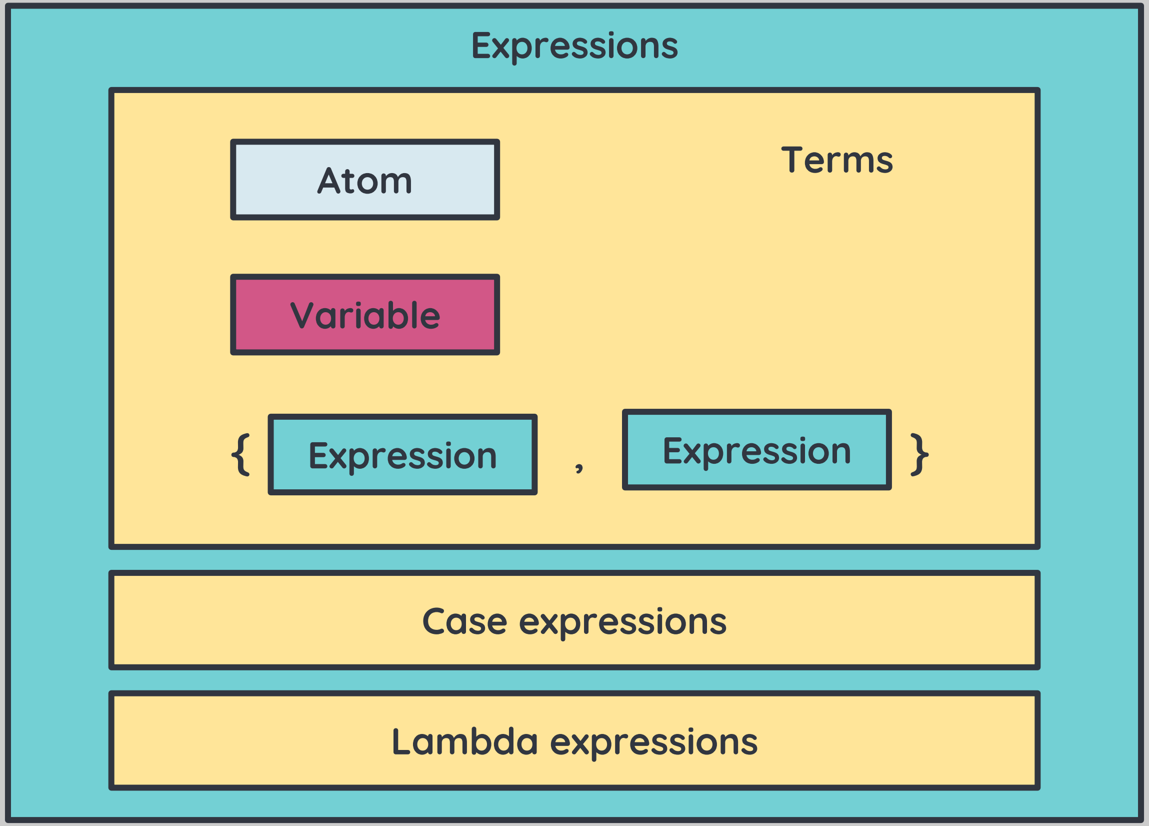
An expression



Example



Terms


Expressions
Atom
Terms
Variable
{ Expression , Expression }
Case expressions
Lambda expressions

Let's evaluate

$$\begin{tikzcd}\textbf{Sequence (terms)} \arrow[r, "\text{evaluation}"] & \textbf{Data structure}\end{tikzcd}$$

Example

```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```

$$\Big\downarrow \text{evaluation}$$

```
{foo, {:bar, :nil}}
```

A pattern

Pattern ::= Atom

Pattern ::= Variable

Pattern ::= Don’t care

Pattern ::= { Pattern , Pattern }

Example

```
x
```

Variable

```
{_, pew}
```

{ Pattern , Pattern }

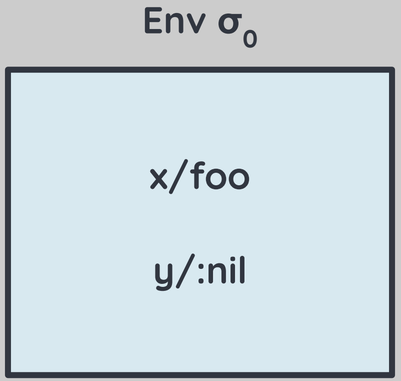
Don’t care Variable

An environment

Contains variables bindings

Initially empty

Immutable: always return new copy



Evaluation: step by step

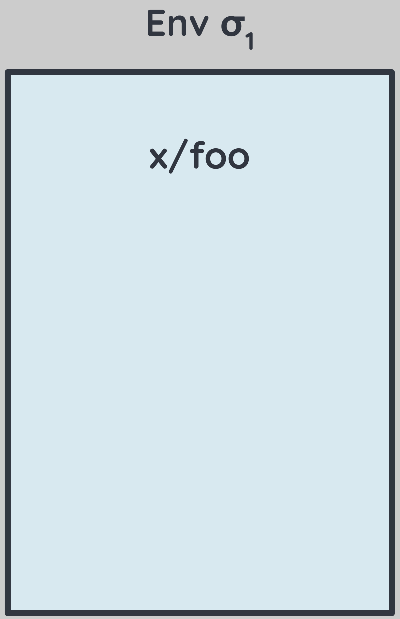
```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```

Env σ_0

empty

Evaluation: step by step

```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```



Evaluation: step by step

```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```

Env σ_2

x/foo

y/:nil

Evaluation: step by step

```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```

Env σ_3

x/foo

y/:nil

z/:bar

Evaluation: step by step

```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```

Env σ_3

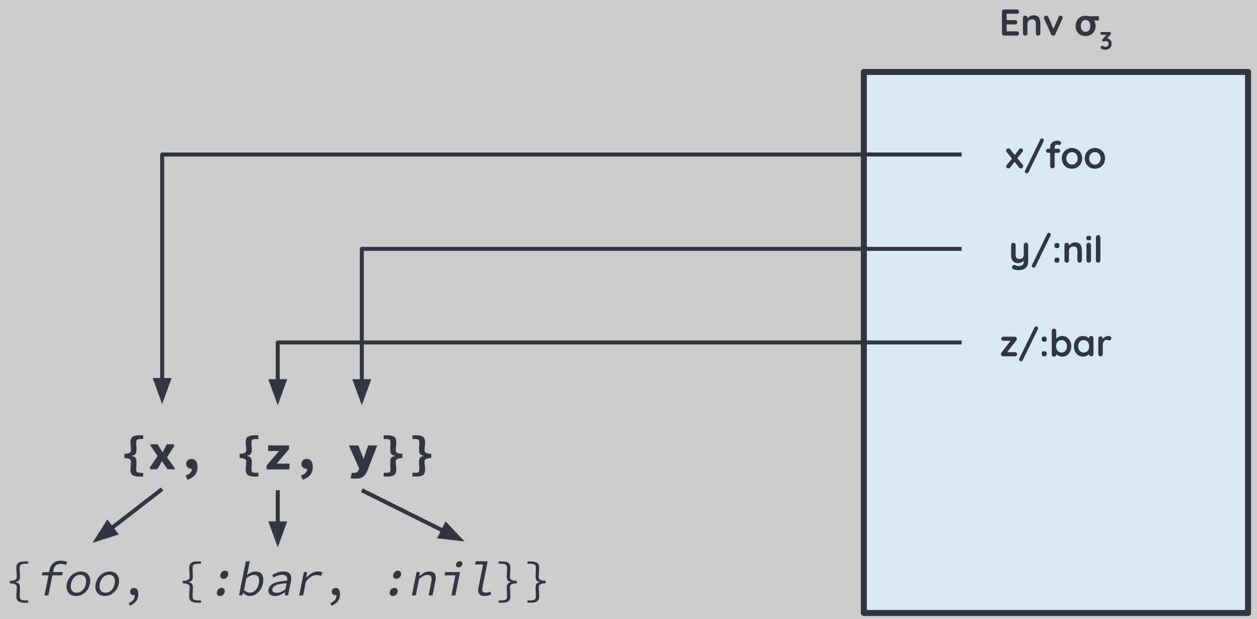
x/foo

y/:nil

z/:bar

Evaluation: step by step

```
x = foo; y = :nil; {z, _} = {:bar, :grk}; {x, {z, y}}
```



Good luck!