1 Overview

- CFGs are a list of rules that describe which sentences are valid within our language.
  - On the left hand side;
    - there will always be a single non-terminal
      - declaration
      - statement
      - expression
  - On the right hand side;
    - there will always be an expression that describes a valid form the non-terminal will take.

---

2 An Example Rule

- We know that a CFG is merely a list of rules. In order to better understand CFGs let us examine the form of one such rule:
  - $A \rightarrow xXy|\epsilon$
- A represents our non-terminal (declaration, statement, or expression)
- $\rightarrow$ is equivalent to "can take the form of"
- x and y are terminal since they are lowercase
- X is non-terminal as it is represented by a uppercase letter
- $|$ is equal to "or" and ϵ is equivalent to nothing or null.

---

3 An Example CFG

- Remember, our first rule is special in that it represents the top level definition of what a valid program is in our language.
  - This is what a abstract CFG looks like:

1.	P	$\rightarrow$	E
2.	E	$\rightarrow$	$E + E$
3.	E	$\rightarrow$	$Identifier$
4.	E	$\rightarrow$	Int

  - There is however a problem with this above CFG. If we look closer at a use case we will see this clearly.

---

4 CFG Ambiguity

- Now we will examine what ambiguity is in the context of CFGs and why it is to be avoided.
  - Let us try to parse a sentence using our grammar to see what is going wrong.
    - Our target sentence will be:
      - $Identifier + Int + Int$
  - We will now apply rules until we reach this sentence as shown below: