CS 152

Programming Paradigms

Language Translation
Lexical Analysis
Context-free Grammars

Today

- Language Translation
- Lexical Analysis
- Context-Free Grammars

Course Learning Outcomes

3. Understand the roles of interpreters, compilers, and virtual machines.
5. Read and produce context-free grammars
6. Write recursive-descent parsers for simple languages, by hand or with a parser generator.

Language Translation

- Translator: a program that accepts other programs and either directly executes them or transforms them into a form suitable for execution.

Language Translation

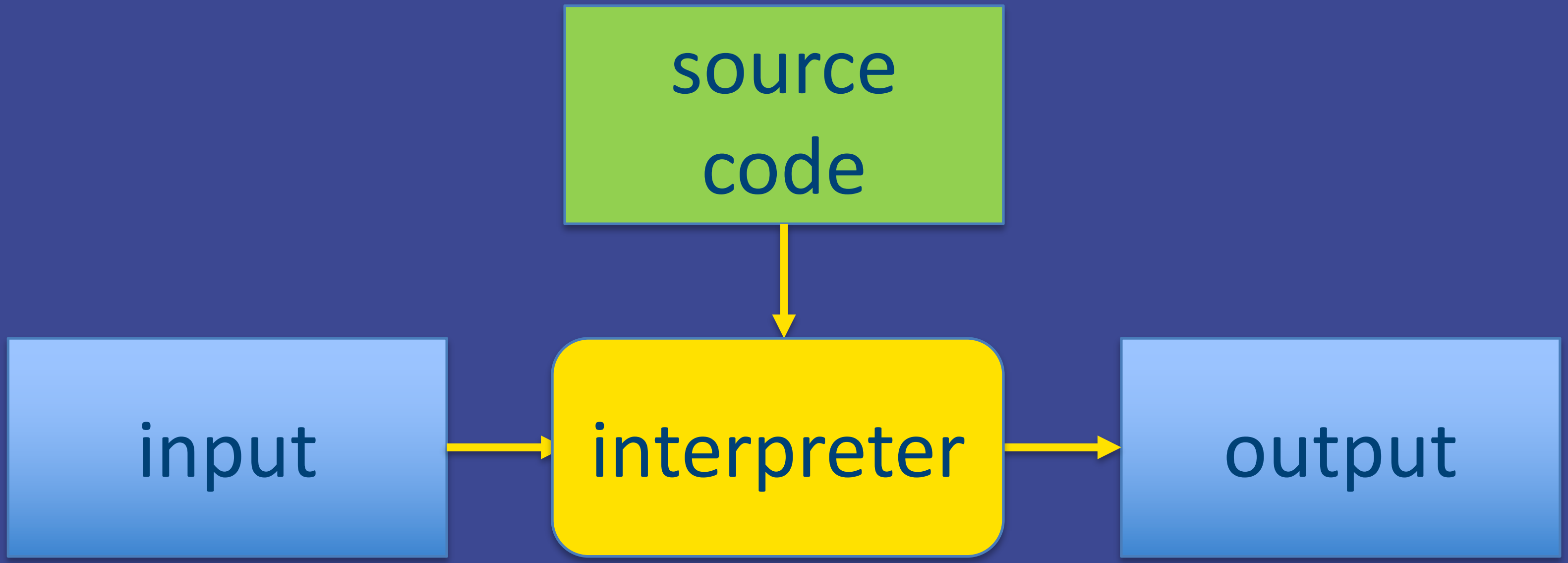
Two major types of 'translators' for a language X:

- Interpreter/evaluator: program written in a language Y that takes a program in X and produces answers (executes it directly)
- Compiler/translator: program written in a language Y that takes a program in X and produces an equivalent program in some target language Z
- In both cases, we call Y the metalanguage

Interpreter

Interpretation is a one-step process

- Both the program and the input are provided to the interpreter, and the output is obtained



Compiler

Compilation requires at least two steps

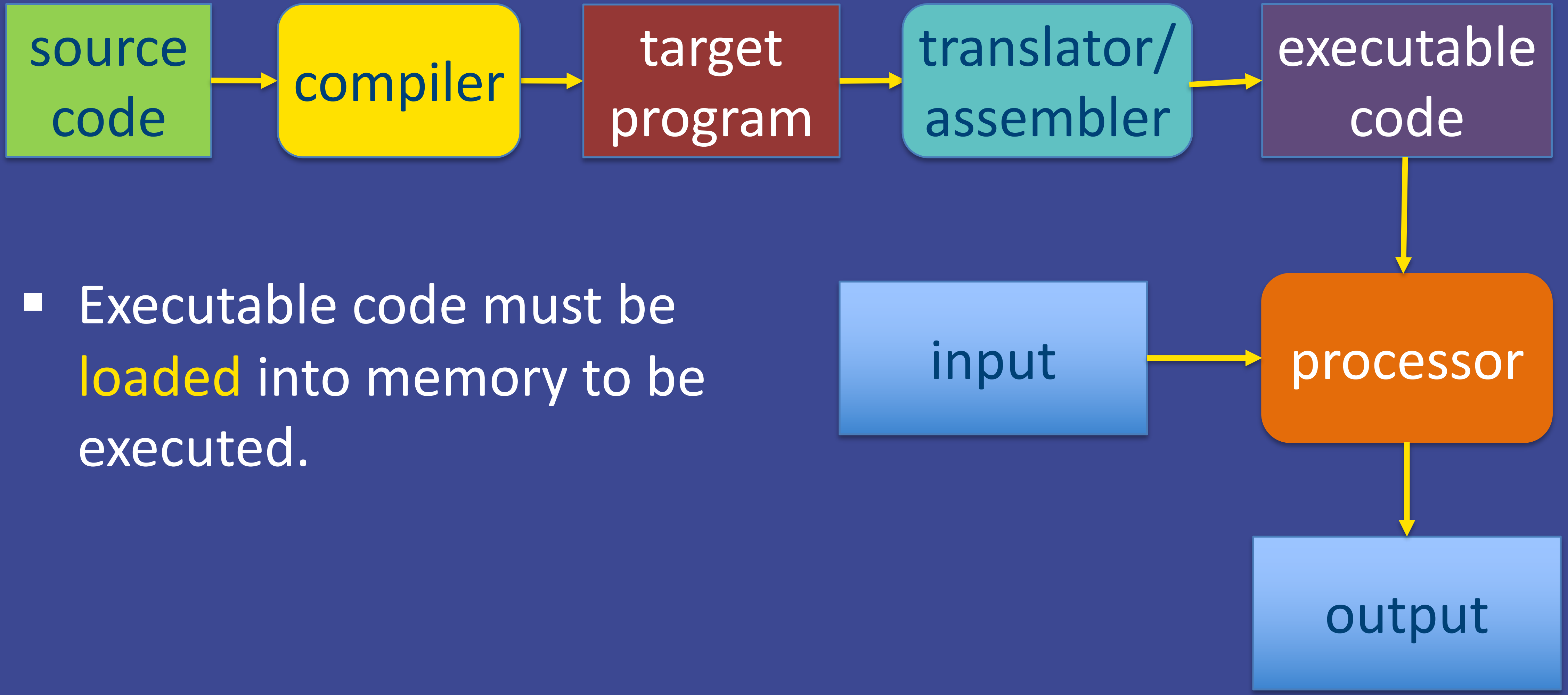
- Source program is input to the compiler
- Target program is output from the compiler
- Target language is often assembly language, so the target program must be translated by an assembler into object code.
- Multiple object programs may be linked into a single executable file.

iClicker

Executable machine code is

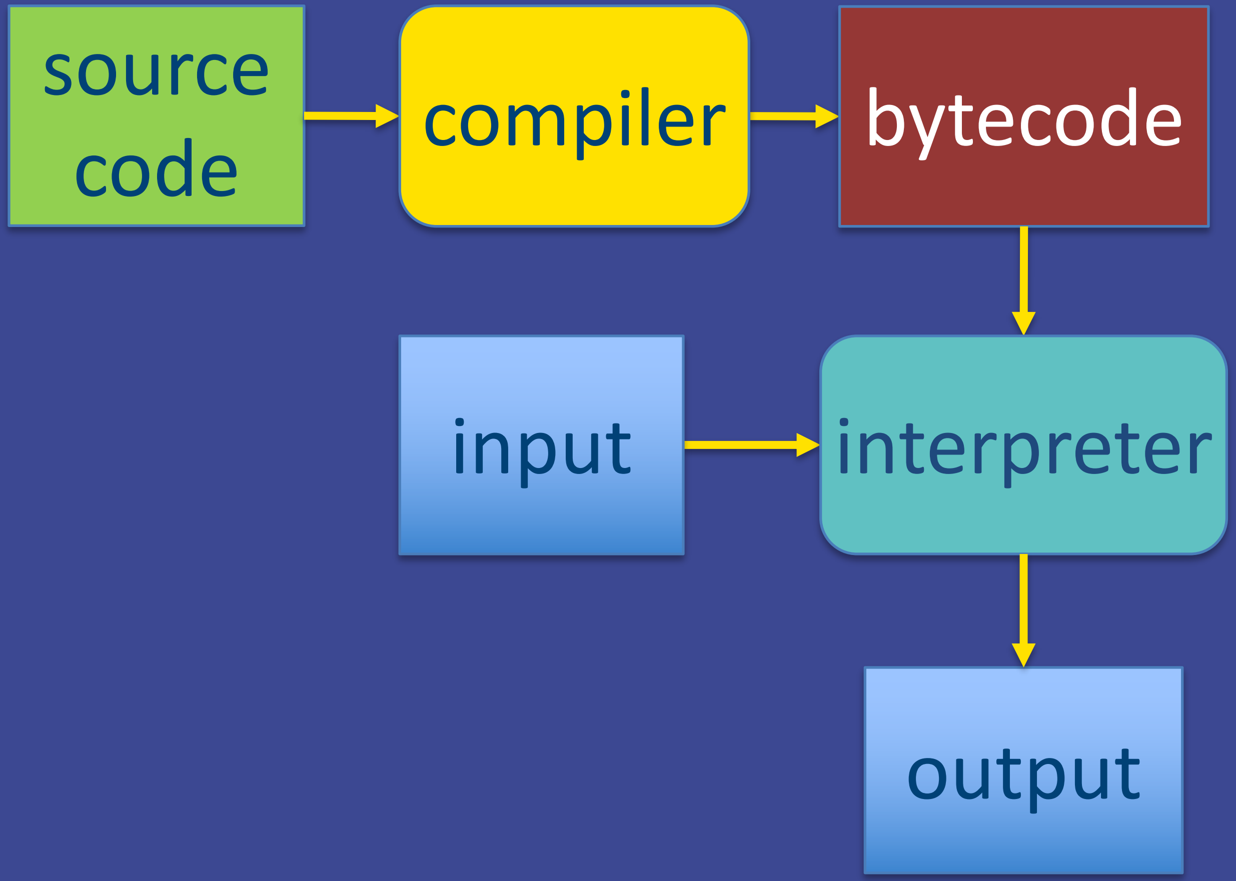
A. Machine independent
B. Machine specific
C. It depends…

Compiler



- Executable code must be loaded into memory to be executed.

Compiler



- Target language may also be bytecode
- Bytecode is machine independent (portable)
- Bytecode is then executed by an interpreter (virtual machine)

iClicker

"Compile once, run everywhere" can be achieved when the target language is:

A. Assembly language
B. Machine language
C. Bytecode

Misconception: Interpreter vs Compiler

- Misconception: there are compiled languages and interpreted languages.
- Whether a programming language is Interpreted or compiled is a feature of a particular implementation, not a feature of the programming language.
- We can write interpreters and compilers for any language!

Lexical Analysis

- Scanning
- Parsing
- Tokens

Syntax vs Lexical Structure

Programming Language Syntax:

- set of rules for the language
- similar to the grammar of a natural language

Lexical Structure:

- structure of the language's words/tokens
- similar to spelling in natural languages

Scanning & Parsing

- Scanner: collects sequences of characters from the input program and forms them into tokens
- Parser: processes the tokens, determining the program's syntactic structure (meaning)

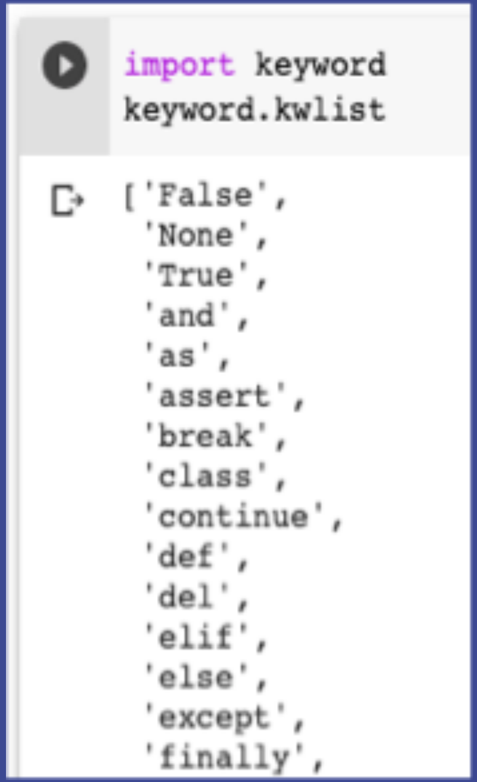
Tokens

Tokens generally fall into several categories:

- Reserved words (or keywords)

```
>>>import keyword
>>>keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Python Examples
Colab notebook



Tokens: Keywords

Tokens generally fall into several categories:

- Reserved words (or keywords)

```
>>> False = 9
SyntaxError: can't assign to keyword
```

```
False = 9

  File "<ipython-input-2-0d1a87f98fe2>", line 1
    False = 9
             ^
SyntaxError: can't assign to keyword
```

Tokens

Tokens generally fall into several categories:

- Reserved words (or keywords)
- Literals or constants

String Literals: 'Hi', "Hello"

Numeric literals:

- Integers: 1, 2, 900
- Floating point literals: 19.99, 0.5, 3.

Tokens

Tokens generally fall into several categories:

- Reserved words (or keywords)
- Literals or constants
- Special symbols:

Operators:

+ - * ** / // % @ << >> & | ^ ~ < > <= >= == !=

Delimiters:

( ) [ ] { } , : . ;

Tokens

Tokens generally fall into several categories:

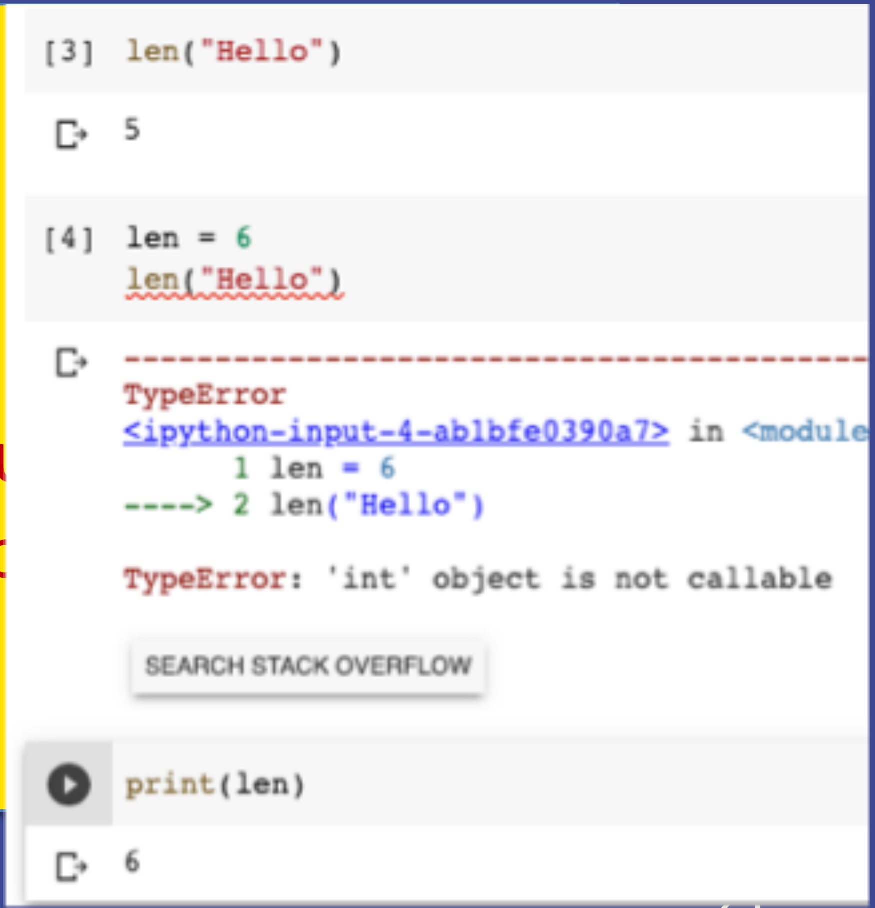
- Reserved words (or keywords)
- Literals or constants
- Special symbols
- Identifiers

result
add, get_value
Account
PI

Tokens: Predefined Identifiers

- Predefined identifiers: identifiers that have been given an initial meaning but are capable of redirection

```
>>> len('Hello')
5
>>> len = 6
>>> len('Hello')
Traceback (most recent call last):  File "<inpu
<module>TypeError: 'int' object is not callab
>>> print(len)
6
```

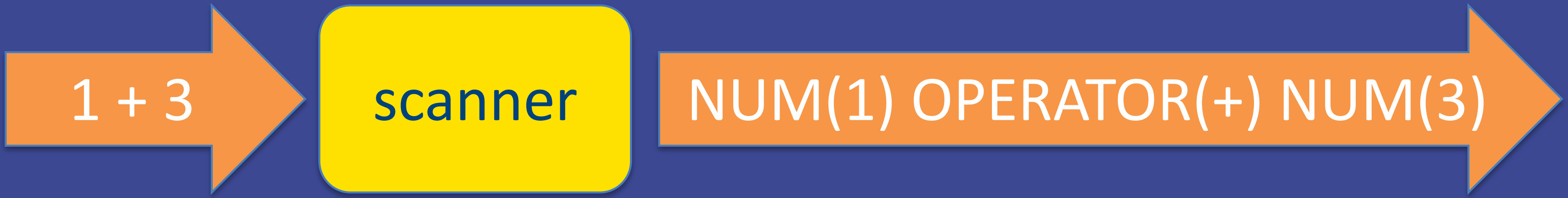


Tokens: Predefined Identifiers

- Predefined identifiers: identifiers that have been given an initial meaning but are capable of redirection

abs, dict, help, min, setattr, all, dir, hex, next, slice, any, divmod, id, object, sorted, ascii, enumerate, input, oct, staticmethod, bin, eval, int, open, str, bool, exec, isinstance, ord, sum, bytearray, filter, issubclass, pow, super, bytes, float, iter, print, tuple, callable, format, len, property, type, chr, frozenset, list, range, vars, classmethod, getattr, locals, repr, zip, compile, globals, map, reversed, __import__, complex, hasattr, max, round, delattr, hash, memoryview, set

Delimiting Tokens



iClicker: Delimiting Tokens

A. Reserved word 'False' and reserved word 'if'
B. Identifier 'Falseif'
C. Syntax error

Principle of Longest Substring

- Principle of longest substring (maximum munch): process of collecting the longest possible string of nonblank characters
- Use whitespace characters as token delimiters (space, tab, new line)

Delimiting Tokens

** power (exponentiation) operator in Python

* multiplication operator in Python

A. OPERATOR (**)
B. OPERATOR (*) OPERATOR(*)
C. Syntax error

Delimiting Tokens

** power (exponentiation) operator in Python

* multiplication operator in Python

Delimiting Tokens

** power (exponentiation) operator in Python

* multiplication operator in Python

A. OPERATOR (**)
B. OPERATOR (*) OPERATOR(*)
C. Syntax error

Delimiting Tokens

** power (exponentiation) operator in Python

* multiplication operator in Python

Fixed Format Language

- Layout is critical
- Specific tokens must be placed in specific columns as on old punched card systems
- Lexical analyzer must know about layout to find tokens

  Fortran:

  Column 1: "C" => comment
  Otherwise:
  Columns 1-5 => label
  Column 6 => continuation
  Columns 7 -72 => statement

Free Form Language

- Only the ordering of tokens is important
- Format has no effect other than satisfying the principle of longest substring
- ALGOL, C, Pascal, Java…
- Lisp, Scheme, etc...

Python?

What about Python?

- Is it fixed format?
- Is it free form?
- Indentation matters: it determines the structure of the program
- This is known as the off-side rule

How do we specify tokens?

- Explicitly list them (if finite):

  Operators

  + - * ** / // % @ << >> & | ^ ~ < > <= >= == !=

- What about identifiers (variable names, etc…)?
- Homework 5 in Haskell (tokenize)
- Another possibility is to use regular expressions and specify a pattern

lex

- Utilities such as lex (or flex) can automatically turn a regular expression description of a language’s tokens into a scanner
- We specify the rules or the pattern (regex) for the set of possible tokens that we want to match

Python lex

```
import lex

# Regular expression rules for  tokens
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_OP = r'\+|-'
t_IDENTIFIER = r'[a-z]+' # only lower case identifiers are supported
def t_INT(t):
    r'\[0-9]+'
    t.value = int(t.value)  # string must be converted to int
    return t
```

Python lex

```
import lex

# Regular expression rules for  tokens
t_LPAREN = r'\('
t_RPAREN = r'\)'
t_OP = r'\+|-'
t_IDENTIFIER = r'[a-z]+' # o
def t_INT(t):
   r'\[0-9]+'
   t.value = int(t.value)  # string must be converted to int
   return t
```

```
# Build the lexer
lexer = lex.lex()
# Send the expression to the lexer
lexer.input(expr)
token = lexer.token() # Get next token
```

Haskell lex vs Alex

- In Haskell, the function *lex* may be used to identify Haskell tokens only:
- It returns a list containing a pair of strings: the first token in the input string and the remainder of the input.

```
 >  lex "first + 5.2"
[("first"," + 5.2")]
 >  lex "5.5+100"
[("5.5","+100")]
```

You are not supposed to use lex or alex in homework 5.

Haskell lex vs Alex

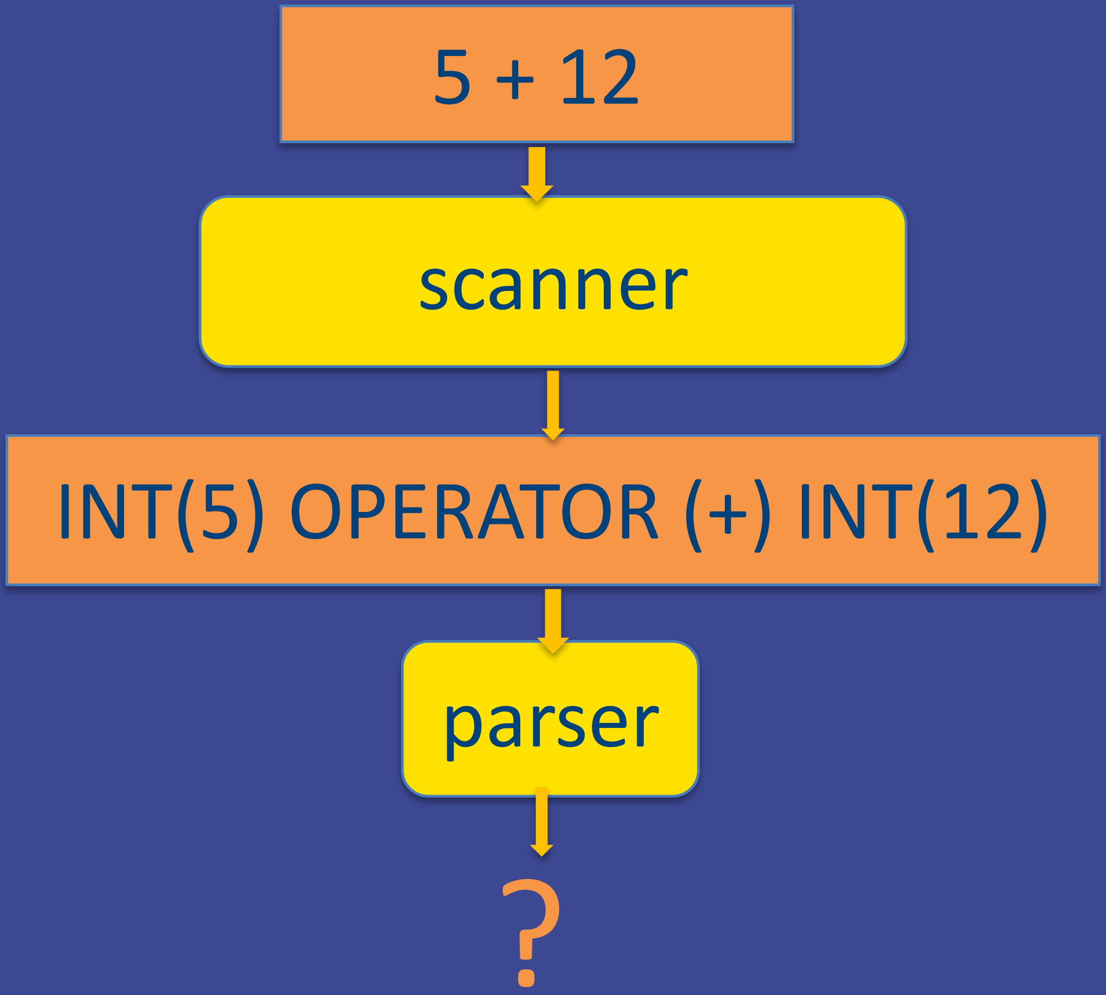
- In Haskell, the package alex may be used to generate lexical analyzers in Haskell, given a description of the tokens in the form of regular expressions.
- It is similar to the tools lex and flex in other languages.

You are not supposed to use lex or alex in homework 5.

Big Picture

- Scanner: collects sequences of characters from the input program and forms them into tokens
- Utilities such as lex can automatically turn a regular expression description of a language's tokens into a scanner

The Big Picture



Language Syntax

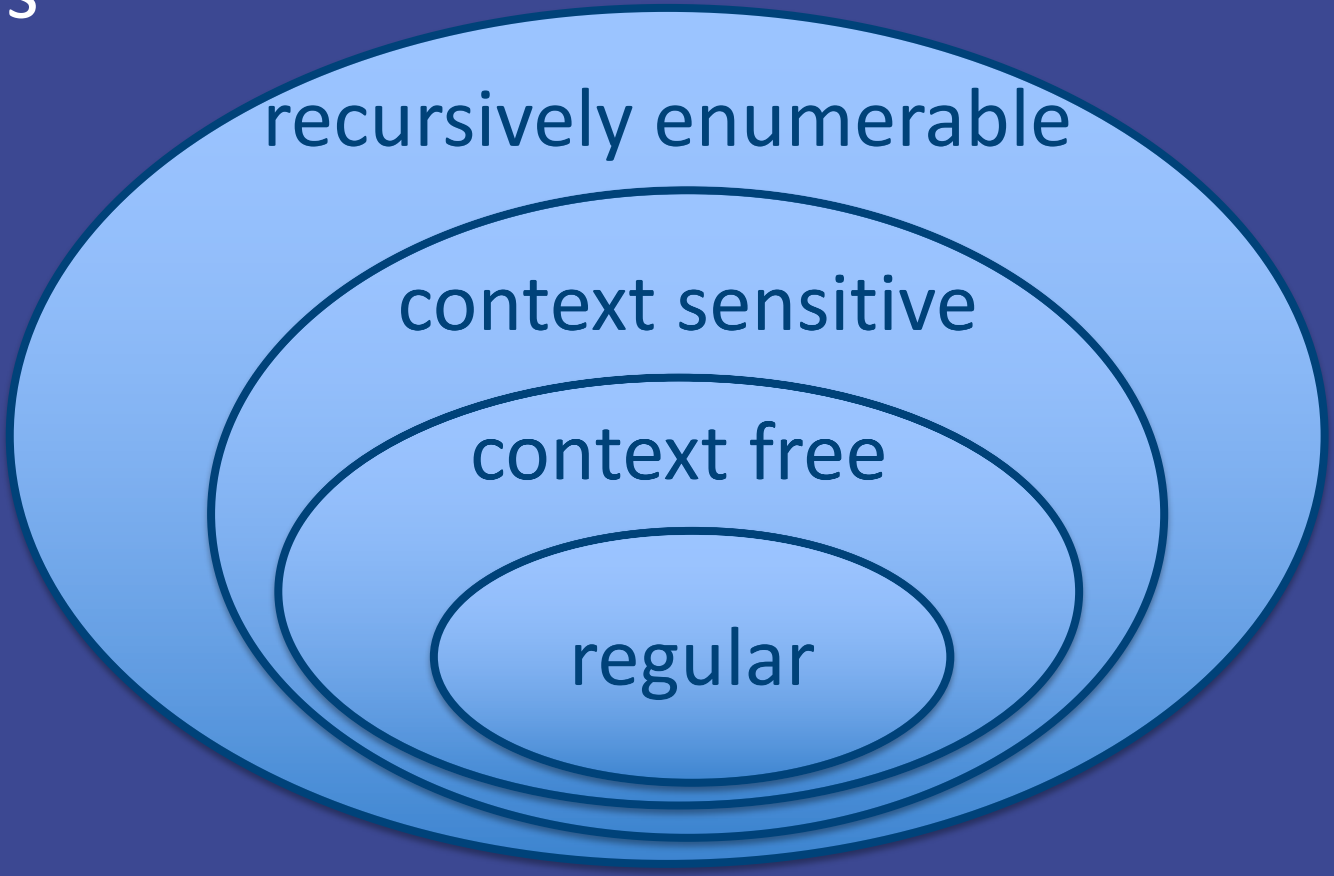
Programming Language Syntax:

- set of rules for the language
- similar to the grammar of a natural language

- Grammar: formal definition of the language's syntax

Background

- 1950s: Noam Chomsky developed the idea of context-free grammars



Background

- John Backus and Peter Naur developed a notational system for describing these grammars, now called Backus-Naur forms, or BNFs
- First used to describe the syntax of Algol60

BNFs

- Three variations of BNF:
  - Original BNF
  - Extended BNF (EBNF)
  - Syntax diagrams

A Simple Grammar Example

1) <sentence> → <noun-phrase> <verb-phrase> .
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

→: is defined as

| : or

A Simple Grammar Example

1) <sentence> → <noun-phrase> <verb-phrase> .
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

metasymbols: symbols used to describe the grammar rules (→, |)

A Simple Grammar Example

1) <sentence> → <noun-phrase> <verb-phrase> .
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Terminals: words/tokens that cannot be broken down further

Terminals: words/tokens (a, the, girl, dog, sees, pets)

A Simple Grammar Example

1) <sentence> → <noun-phrase> <verb-phrase> .
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Non-terminals: can be broken down

Non-terminals are enclosed in angle brackets <sentence>, <noun>,…

A Simple Grammar Example

- words/tokens may be enclosed in quotes to differentiate them from metasymbols

1) <sentence> → <noun-phrase> <verb-phrase> "."
2) <noun-phrase> → <article noun>
3) <article> → "a" | "the"
4) <noun> → "girl" | "dog"
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → "sees" | "pets"

A Simple Grammar Example

- in pure text (no formatting)

1) <sentence> ::= <noun-phrase> <verb-phrase> "."
2) <noun-phrase> ::= <article> <noun>
3) <article> ::= "a" | "the"
4) <noun> ::= "girl" | "dog"
5) <verb-phrase> ::= <verb> <noun-phrase>
6) <verb> ::= "sees" | "pets"

A Simple Grammar Example

- In BNF ISO Standard

1) sentence = noun-phrase , verb-phrase , "." ;
2) noun-phrase = article , noun ;
3) article = "a" | "the" ;
4) noun = "girl" | "dog" ;
5) verb-phrase = verb , noun-phrase ;
6) verb = "sees" | "pets" ;

Production

- Production: another name for grammar rules
  - Ex: <noun-phrase> → <article> <noun>

1) <sentence> → <noun-phrase> <verb-phrase> .
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Start Symbol

Start symbol: a nonterminal representing the entire top-level phrase being defined

Start symbol?

<sentence>

1) <sentence> → <noun-phrase> <verb-phrase> .
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation

- Derivation: the process of building sentences by beginning with the start symbol and replacing left-hand sides by choices of right-hand sides in the rules

Derivation Example

Start symbol?

<sentence>

1) <sentence> → <noun-phrase> <verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>

⇒ <noun-phrase><verb-phrase>.

1) <sentence>→<noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>

⇒ <noun-phrase><verb-phrase> .

⇒ <article><noun><verb-phrase> .

1) <sentence>→<noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>

⇒ <noun-phrase><verb-phrase> .

⇒ <article><noun><verb-phrase>.

⇒ the <noun><verb-phrase>.

1) <sentence> → <noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>

⇒ <noun-phrase><verb-phrase> .

⇒ <article><noun><verb-phrase>.

⇒ the <noun><verb-phrase>.

⇒ the girl <verb-phrase>.

1) <sentence>→<noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>
⇒ <noun-phrase><verb-phrase> .
⇒ <article><noun><verb-phrase>.
⇒ the <noun><verb-phrase>.
⇒ the girl <verb-phrase>.
⇒ the girl <verb> <noun-phrase>.

1) <sentence> → <noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>

⇒ <noun-phrase><verb-phrase> .

⇒ <article><noun><verb-phrase>.

⇒ the <noun><verb-phrase>.

⇒ the girl <verb-phrase>.

⇒ the girl <verb> <noun-phrase>.

⇒ the girl sees <noun-phrase>.

1) <sentence>→<noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>

⇒ <noun-phrase><verb-phrase> .

⇒ <article><noun><verb-phrase> .

⇒ the <noun><verb-phrase>.

⇒ the girl <verb-phrase>.

⇒ the girl <verb> <noun-phrase>.

⇒ the girl sees <noun-phrase>.

⇒ the girl sees <article><noun>.

1) <sentence> → <noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>
⇒ <noun-phrase><verb-phrase> .
⇒ <article><noun><verb-phrase>.
⇒ the <noun><verb-phrase>.
⇒ the girl <verb-phrase>.
⇒ the girl <verb> <noun-phrase>.
⇒ the girl sees <noun-phrase>.
⇒ the girl sees <article><noun>.
⇒ the girl sees a <noun> .

1) <sentence>→<noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Derivation Example

<sentence>
⇒ <noun-phrase><verb-phrase> .
⇒ <article><noun><verb-phrase>.
⇒ the <noun><verb-phrase>.
⇒ the girl <verb-phrase>.
⇒ the girl <verb> <noun-phrase>.
⇒ the girl sees <noun-phrase>.
⇒ the girl sees <article><noun>.
⇒ the girl sees a <noun> .
⇒ the girl sees a dog .

1) <sentence> → <noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Other Possible Derivations?

<sentence>
⇒ <noun-phrase><verb-phrase>.
⇒ <article><noun><verb-phrase>.
⇒ the <noun><verb-phrase>.
⇒ the dog <verb-phrase>.
⇒ the dog <verb><noun-phrase>.
⇒ the dog pets <noun-phrase>.
⇒ the dog pets <article><noun>.
⇒ the dog pets a <noun>.
⇒ the dog pets a girl.

1) <sentence>→<noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Other Possible Derivations?

<sentence>
⇒ <noun-phrase><verb-phrase>.
⇒ <noun-phrase><verb><noun-phrase>.
⇒ <noun-phrase>pets<noun-phrase>.
⇒ <noun-phrase>pets <article><noun>.
⇒ <noun-phrase> pets the <noun> .
⇒ <noun-phrase> pets the girl .
⇒ <article> <noun> pets the girl .
⇒ a <noun> pets the girl .
⇒ a dog pets the girl .

1) <sentence> → <noun-phrase><verb-phrase>.
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

Limitations?

- Do all legal sentences make sense?
- Capitalization? Spaces?

1) <sentence> → <noun-phrase> <verb-phrase> .
2) <noun-phrase> → <article> <noun>
3) <article> → a | the
4) <noun> → girl | dog
5) <verb-phrase> → <verb> <noun-phrase>
6) <verb> → sees | pets

BNF?

- Backus-Naur form: uses only the metasymbols → (or ::=) and |
- No repetitions (*, +)

1)<sentence>→<noun-phrase><verb-phrase>.
2)<noun-phrase> → <article> <noun>
3)<article> → a | the
4)<noun> → girl | dog
5)<verb-phrase> → <verb> <noun-phrase>
6)<verb> → sees | pets

Context-free?

- Each production rule has a single non-terminal on the left, then a → metasymbol, followed by a sequence of terminals/tokens or other non-terminals on the right
- There is no context under which only certain replacements can occur
- Typically there are as many productions in a context-free grammar as there are non-terminals
- Terminals never appear on the left hand side of a rule