··Topics:

- Difference between Semantics and Syntax
  - different programming languages often provide features with very similar semantics but very different syntax.
  - There are some very efficient and elegant algorithms that a compiler and interpreter can use to discover they syntactic structure(but not the semantics) of a computer program.
  - Syntax is a form that how we specify the structural rules of a computer program, and how a compiler identifies the structure of a given input program.-àwhat a language is
    - Two tasks: specify how to generate valid program and which recognize program structure.
  - Semantics is a meaning what a language does.
    - Static semantics: what happen at compile time
    - Dynamic semantics: what happens at runtime
      - Divided by 0; null pointer; stack overflow; out of memory; bounds checking;
- Difference between assembler & compiler
  - Assembler 汇编器：translate from mnemonics(帮助记忆的词句) to machine language became the job of a systems program. (one-to-one correspondence between mnemonics and machine language instructions.)
  - Compiler translate high-level language to assembly or machine language
  - Compiler is more complicated than assembler because the ono-to-one correspondence between source and target operation no longer exists.
  - Assembly language programs run faster than compiler.
  - Assembly language is more difficult to learn than high-level language.
- Why are there different programming languages?
  - Evolution. CS is young.
  - Special purpose
  - Personal preference
  - What makes a language successgul?
    - Expressive power: Turing complete 图灵完备 each can be used to implement arbitrary alogorithm
    - Ease of Use for the Novice
    - Ease of implementation
    - Standardization
    - Open source
    - Excellent compiler
- BNFs (how to write, + what they are used for). Writing a parse tree.
  - What is a parse tree?
  - Kleene plus it indicates one or more instances of the symbol or groups of symbols in front of it.
  - Kleene star
- Compilation:
  - Know all the steps we have discussed
  - describe the purpose of each stage and its input and output
  - Example optimizations
  - Examples of what can't be checked until run time (dynamic semantics) and how it is accomplished.
  - Interpreting vs. compiling. What's the difference? pros/cons?
- List the paradigms we have discussed and their characteristics. + example language

- Declarative: what the computer is to do (more programmer's point of view)
  Interpreter/declare SQL
  Functional, Lisp, Haskell computational model based on the recursive definition of function. It is close to mathematical specification; What is the computed;
  dataflow, Id
  logic, logical statement that describe properties + fact and rules; Prolo;
  template-based XSLT
- Imperative: how computer should do it (more implementer's point of view):
  compiler; step by stop!! Requires an ordered list of instructions for execution
  Object-oriented; java Programming by modeling objects and solving problems using their relationships
  von Neumann（Procedural）, c;
  scripting. JavaScript, PHP, Python
  Procedural: is part of imperative; introduce notation of functions

- Type conversion vs type coercion
  -
  - Javascript will quietly convert that value to the type it needs, using a set of rules that often aren't what you what or expect. String(), Number();Boolean();
- Static vs dynamic scoping
  - Static scoping: the scope of an identifier is determined by its location in code; where in code am I defined?
  - Dynamic scoping: in determined by a sequence of calls that has led to use of an identifier. Who called me?
- Hoisting
  - Hoisting: declaration of variables and functions are on top;
  - closure
- Var vs let
- Closure: the function is closes over some local bindings
  - Being able to reference in a specific instance of a local binding in an enclosing scope is called.
- Temporal Dead Zone
  - A point between entering scope and declaring a variable, in which that variable cannot be accessed.
- Some questions over JavaScript from the readings.
- Pass by Value vs. Pass by Reference
  - Copy a value in stack;
  - Reference is return address! Point to other memory locations.
  - Javascript always pass by value
- Objects v primitives
  - Primitive pass by value has no properties. Primitive copy the location and parse to a new place; number string Boolean function symbol undefined。
  - Objects are not passed by reference, they are passed by value.
  - Difference in copy;
- Call stack
  - The place where the computer stores this context; Every time a function is called, the current context is stored on top of this "stack". When a function returns, it removes the top context from the stack and uses that to continue execution.
  - If stack memory is too big, out of stack space

Typeof: Object number Boolean function undefined symbol string (7 things)
Non-primitives: objects
New Object is in heap.

Parsing and scanning -à translate the program into an equivalent program in the target program.
Scanner (lexical analysis): remove comment; reduce the number of individual items; save

identifiers and tags token
Parsing (syntax analysis): obtain the token of input program, assemble the token together into a syntax tree, and passes the tree to the later phases of the compiler. -àa language recognizer.
Parsing: LL left-to-right left-most derivation. Also called Top-down or predictive parses

Machine language ==object code

Pros and Cons
Interpreter: stays around for the execution of the applications; great flexibility and better diagnostics (error message) than compiler but slow execution
Preprocessor: most interpreted languages employ an initial translator that remove comments and white space and group characters together into tokens.
Compiler: better performance, fast execution, fewer runtime errors, optimization, cannot be modified by users.
Linker: merge the appropriate library routines into the final program
Compiler -> assembler -> machine language? Because assembler is easy to debug and read.

Article about Java and C
https://www.ibm.com/developerworks/library/j-jtp09275/

RadioLab: Word
Importance of knowing different paradigms & languages?
What are potential errors we can check for during Semantic Analysis?
Progress compile:
Step 1: scanning 2: Parsing 3:Semantic Analysis & intermediate code generation(Meaning and things not captured by syntax tree) 4: Machine Independent code Improvement 5 target code generation 6 Target code improvement(optimization)
https://downloads.haskell.org/~ghc/7.10.3/docs/html/users_guide/options-optimise.html
https://courses.cs.washington.edu/courses/cse401/06sp/codegen.pdf
MDN https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval

Paradigm: a typical example or pattern of something, model
Programming paradigm 编程范例 https://www.imooc.com/article/14330

Strict mode:
it would usually allow, such as not used let when declaring, issues with functions that are not called as methods, multiple parameters of the same name, etc

declaration function:

short-circuiting of logical operators:

binding == variables

Object.keys();

DOM
All changes are made through the Document Object Model (DOM). What is the Document Object Model?
• The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page. [. . .]