Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus
You can program without programming languages... if you really want

xkcd
Computer Architecture in One Slide

<table>
<thead>
<tr>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>01100111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
</tr>
<tr>
<td>126</td>
</tr>
<tr>
<td>35</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>987</td>
</tr>
<tr>
<td>1024</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

Op Code | Data 1 | Data 2 |
--------|--------|--------|
        |        |        |
        |        |        |

Program Counter
You can program without programming languages... if you really want

Altair 8800
1974
You can program without programming languages... if you really want

Instruction tape for Harvard Mark I
~1944
Assembly code makes instructions more human-readable
If we can turn text into binaries, why not easier-to-write text?

Rear Admiral Grace Hopper (1906-1992)

COBOL (1959)
Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus
There are different ways of translating a programming language

- **Source Code**
  - Compiler
  - Binary/Assembly
    - Ex.: C, C++

- **Source Code**
  - Interpreter
    - Ex.: Python

- **Source Code**
  - Compiler
  - Bytecode
    - VM
    - Ex.: Java
Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus
All programming languages are the same... in a deep sense

“Turing completeness”

But the choice of language still matters in a very real sense—languages are tools!
Programming Language =

Syntax  What programs *look like*

+  

Semantics  What programs *mean*
Syntax vs. semantics: Python

def func():
    return 5 + "hello"

File "main.py", line 1
    def func():
        ^
SyntaxError: invalid syntax

File "main.py", line 2, in func
    return 5 + "hello"
TypeError: unsupported operand type(s) for +: 'int' and 'str'
Syntax vs. semantics: OCaml

Let's consider the following OCaml function:

```ocaml
let func () = 5 + "hello"
```

This code is syntactically valid, but semantically it contains a type error. The expression `"hello"` is of type `string`, but an expression is expected of type `int`. The compiler will report an error at Line 5, characters 18-25:

Error: This expression has type string but an expression was expected of type int
Semantics =

Static

Analyzed at compile time

+ Where do we check types?

Dynamic

Happens at run time
We can divide programming languages by whether they have *static* or *dynamic* types

- Static languages: types checked at compile time: *no type errors* at runtime
  
- Dynamic languages: types checked at run time, can have type errors
  
- (Weakly typed languages): types checked at compile time, but can be avoided, resulting in unexpected behavior or type errors at run time
We can also divide programming languages based on *paradigm* (how you think about programming)

- Imperative: *tell computer what to do*
- Functional: *describe the computation mathematically*
- Object-oriented: *objects perform computation and carry data*
- Scripting
- Relational
- Domain-specific
- Logic
<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imperative</td>
<td>Typescript, Pascal</td>
<td>Python, Javascript</td>
</tr>
<tr>
<td>Functional</td>
<td>Haskell, OCaml</td>
<td>Scheme, Racket</td>
</tr>
<tr>
<td>Object-oriented</td>
<td>C++, C#, Java</td>
<td></td>
</tr>
</tbody>
</table>
Knowing the right paradigm to use can make programming easier

Task: Sort a linked list (using merge sort)
Knowing about the language and how it’s translated can help you write faster code

<table>
<thead>
<tr>
<th>Language</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0.007</td>
</tr>
<tr>
<td>Python</td>
<td>6.43</td>
</tr>
<tr>
<td>OCaml (bytecode)</td>
<td>0.014</td>
</tr>
<tr>
<td>OCaml (native)</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Merge sort, 10,000 elements
Knowing about the language and how it’s translated can help you write faster code.

![Bar chart showing time in milliseconds for various languages]

- C: 7 ms
- Python: 6430 ms
- OCaml (bytecode): 14 ms
- OCaml (native): 10 ms

Merge sort, 10,000 elements
Type systems can express different levels of guarantees

• C
  node *mergesort(node *list)
  • Takes a pointer to a node and returns a pointer to a node.

• OCaml
  mergesort : int list -> int list
  • Takes an integer list and returns an integer list.

• Haskell
  mergesort :: IO ([int] -> [int])
  • Takes an integer list, returns an integer list and performs I/O (e.g., printing).

• Coq
  mergesort : forall (l1 : list int), exists (l2: int list),
  \[Sorted l2 \land \text{Permutation } l1 \ l2\]
  • Takes an integer list and returns a sorted permutation of it.
Different languages are up to different tasks
Outline

1. Programming Languages
2. Translators: Compilers and Interpreters
3. Types of Programming Languages
4. Syllabus
Course Goals

• Learn to evaluate and discuss programming languages
  • Learn the lingo (impressing people with jargon isn’t the point, but is a side effect)
    \[ \Gamma \vdash e_1 : \tau_1 \quad \Gamma \vdash e_2 : \tau_2 \]
    \[ \Gamma \vdash (e_1, e_2) : \tau_1 \times \tau_2 \]

• Reason precisely about what programs mean
  • Inject mathematical rigor into programming

• Become a creator of PLs, not just a consumer
Sections/Attendance

• Section 01: In-person
  • Attendance not recorded, but attendance/participation may be used to “break ties”

• Section 02: Online
  • Lecture videos will be posted to Blackboard after each lecture
Course Staff

• Instructor: Stefan Muller
  • Office Hours: Mon., 11am-12pm (Online – link to come)
    Thur., 2-3pm (SB 218E)

• TA: Xincheng Yang
  • Office Hours: Tue., 2-4pm (Online – appt. link to come)
    Wed., 2-4pm (SB 004)
Course Website: [http://cs.iit.edu/~cs440/](http://cs.iit.edu/~cs440/)

Important info, notes, etc.

---

**CS440: Programming Languages and Translators, Spring 2023**

**Instructor:** Stefan Muller, smuller2@IIT.edu

**Office Hours:** TBA

**TA:** TBA

**Lectures:**
- Section 01: Tue, Thu 10:00-11:15 AM, SB 194
- Section 02: Online only

---

**Schedule**

Note: this schedule is tentative and subject to change.

For the readings posted:

- "PDDB" = "Purple Dragon Book" (Ari et al.)
- "FPF" = "Functional Programming in OCaml" (link below)
- "TAPL" = "Types and Programming Languages" (Harper)
- "PFPL" = "Practical Foundations for Programming Languages" (Harper)

<table>
<thead>
<tr>
<th>January</th>
<th>Topic</th>
<th>Readings Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Intro</td>
<td>Languages and course overview</td>
</tr>
<tr>
<td>12</td>
<td>OCaml</td>
<td>Compiler structure, interpreters, OCaml</td>
</tr>
<tr>
<td>17</td>
<td>OCaml</td>
<td>OCaml evaluation, types, expressions</td>
</tr>
<tr>
<td>19</td>
<td>OCaml</td>
<td>Functions and recursion</td>
</tr>
<tr>
<td>24</td>
<td>OCaml</td>
<td>More on types</td>
</tr>
<tr>
<td>26</td>
<td>OCaml</td>
<td>Lists and tail recursion</td>
</tr>
<tr>
<td>31</td>
<td>OCaml</td>
<td>Records and algebraic data types</td>
</tr>
<tr>
<td>February</td>
<td>OCaml</td>
<td>Higher-order functions</td>
</tr>
<tr>
<td>2</td>
<td>OCaml</td>
<td>Higher-order functions</td>
</tr>
<tr>
<td>9</td>
<td>OCaml</td>
<td>Side effects</td>
</tr>
<tr>
<td>14</td>
<td>OCaml</td>
<td>Building an interpreter</td>
</tr>
<tr>
<td>16</td>
<td>OCaml</td>
<td>Closures</td>
</tr>
</tbody>
</table>
Other ways to get help

- **Discord**: IIT CS server, #cs440 channel
  - If you’re not on it, we’ll send an invitation

- **Academic Resource Center (ARC)**: [www.iit.edu/arc](http://www.iit.edu/arc)
  - FREE subject matter tutoring and academic coaching

<table>
<thead>
<tr>
<th></th>
<th>Discord</th>
<th>Office Hours</th>
<th>Email</th>
<th>ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td>General questions about lectures, logistics, etc.</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>General discussion, clarifications, about HW questions</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specific questions about your HW answers</td>
<td></td>
<td>✔️</td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>More in-depth personal tutoring</td>
<td>✔️</td>
<td></td>
<td></td>
<td>✔️</td>
</tr>
<tr>
<td>Personal matters (accommodations, other requests, etc.)</td>
<td></td>
<td></td>
<td>✔️</td>
<td></td>
</tr>
</tbody>
</table>
Collaboration and Academic Honesty

• Discussing general concepts is encouraged
• Discussing broad strategies for doing lab tasks is OK – don’t discuss actual answers or code
  • If, after your discussion, you don’t take any notes/pictures and write up your code/solutions by yourself, you’re probably OK
  • Cite collaborators and any other resources in your write-up

• Not allowed:
  • Working together
  • Sharing answers
  • Looking for answers on the internet

This is the short version: read the details on the course website
(Tentative) Schedule

- Intro (1 week – you are here)
- Learn OCaml (~4 weeks)
- Interpreters (~2 weeks)
- Midterm
- Type checking (~2 weeks)
- Spring break
- Formal semantics (~2 weeks)
- Formal type systems (~2 weeks)
- Other topics and wrap-up (~3 weeks)
Labs/Projects/Homeworks/Problem sets

• 6-7 homeworks, ~2 weeks each
  • Lab 0 Out ~Thursday, Due 1/26
• Written and programming
• Work individually

Late Days:
• 7 per student, extend deadline 24 hours
• No more than 2 per assignment
• If no more late days, 10% late penalty per day
• No work accepted >48 hours late
Exams

• Midterm (tentatively Mar. 2)
• Final (finals week)

• Details TBA

• (No using late days, sorry)
Grading

• 50% Homeworks
• 20% Midterm
• 30% Final

<table>
<thead>
<tr>
<th>Score Range</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=90</td>
<td>A</td>
</tr>
<tr>
<td>80-90</td>
<td>B</td>
</tr>
<tr>
<td>70-80</td>
<td>C</td>
</tr>
<tr>
<td>60-70</td>
<td>D</td>
</tr>
<tr>
<td>&lt;60</td>
<td>E</td>
</tr>
</tbody>
</table>
Textbooks

On Compilers/interpreters:

• Appel. *Modern Compiler Implementation in ML*
• Nystrom. *Crafting Interpreters*

For more math-y details:

• Pierce. *Types and Programming Languages*
• Harper. *Practical Foundations for Programming Languages*
For Thursday: Bring laptops if you can!