La lezione di oggi è consistita in un ripasso generale, ma sono state accennate alcune cose interessanti, che riporto
in questo documento, senza un ordine particolare.

Il back-end del compilatore effettua anche ottimizzazioni machine-dependent:
Ad esempio, ***common subexpression elimination***:
La seguente porzione di codice contiene la ripetizione dell'operazione `z * π`, che è onerosa:

```
x := z * π * 2
...
y := z * π * 5
```

Diventa:

```
a := z * π
x := a * 2
...
y := a * 5
```

Ottimizzazione ***definition use***: si basa su un grafo.

https://en.m.wikipedia.org/wiki/Reaching_definition [?]

Quando si scrive un compilatore, il goal è quello di avere il massimo di informazioni estraibili in fase di analisi sintattica, sia che questa sia top-down che bottom-up.

Ho due modi differenti per generare il codice intermedio:
Costruisco l'AST durante l'analisi sintattica → dall'AST genero il codice intermedio.
Arricchisco la grammatica con azioni semantiche, ottenendo un SDD → genero direttamente in fase di analisi sintattica.

Il più famoso three-address code si chiama SSA (Single State Assignment). Che agevola le operazioni di utilizzazioni

Pyhton e linguaggi di ultima generazione: l'indentazione ha un significato semantico, tipo quello della parentesi:
solo che la parentesi la vedo, la tabulazione no.

Accennato: scannerless parsing.