Explicit type conversion: (int)x
≈ cast

Implicit type conversion char y='a'
// convert y to a value of 97
int x = 10;
float z = x+1.0; //x is implicitly
convert to float

(3.14159+5) is capability type

(int)95.7 : type conversion

in OCamel, unification ...→ inference

in C, declare an array w structelem
→ Ortho

C++, use of virtual func → Dyn. typing

Best describe OCaml : Dyn.. Structural
{Strongly name S/equi

[|1;2;3|] → an array w 3 int ele 1,2 & 3

In OCaml, !y is the value (content) accessed by the ref variable y

Option types programmer to specify a value valid/invalid → Ⓣ

- C++ denotational and abstraction-based → Ⓣ
- Programming language highly ortho ... less ortho → Ⓕ
- Real number type, scalar type Ⓕ
- Large number coercionss between decrease ease of use and understandability → Ⓣ
- Functional language make extensive use of side effects → Ⓕ
- Functional language manipulated same mechanism manipulate data Ⓣ
- lambda calculus C++ Java Ⓣ
- Most functional langue do NOT support Ⓕ
- Garbage collection essential feature → Ⓣ

Why does OCaml provide separate
→ OCaml does not Support implicit casting an adding an integer with a float. Hence, there's not automatic conversion in an expression where Ocaml requires that there will be two seperate arimetic operators.

Briefly explain the difference btw physical and structural equality of values in OCaml? → Physical equality checks if there are two data structures that have the same pointer in memory. The structural eq'l check the fields of two values and checks if the fields are equal. Physical eq. uses "=", structural uses "==".

Name >< Structural of types →
→ Name checks if two things have the same name; Structural check for the same of two structures. Name would be similar to physical equality and to structural equivalence, there's structural equality that is similar.

Ada: Celcius ↔ Fahrenheit: capability is useful because the program may have capable code that converts value of integer and float. Celcius from/to Fahrenheit to

int + int = int ; int+double=double

:= operator to assign to references
! deferences to get out the contents

- LL parser top-down → Ⓣ LR comes LL Ⓣ
- Stage reads a stream of char makes a stream of token → Scanner
- Stage determines the meaning of a program, errors and declaration before use → Semo
- Grammar ambigous, type grammar → None of LL LR, LALR, SLR
- Grammar recursive descent → LL
- If a grammar left-recursive → LR, LALR, SLR
- Module 2- Ada → "end marker"
- Front end → Parser
- High level intermediate form → Abstract
- On modern machines, assembly lang > compiler Ⓕ
- BNF enables Algol-60 → Ⓣ
- Operator precedence "higher" → Ⓕ
- Scanner peek Ⓣ
- Epsilon LR > LL Ⓕ
- Top down A→α T∈First(α) → Ⓣ
- S-attributed synthesized, inherited → Ⓕ
- recursive desent incorp semantic routes Ⓕ
- frontend GCC RTL Ⓕ
- machine dependant number of reg Ⓕ
- which is NOT regular expression generate tokens of a programming language → recursion
- DO Loop in FORTRAN → Variable need not Spaces...
- LL top-down → Ⓣ

- array allocation → local array static / runtime dyn
- Smart pointers → reference counts
- int my_arry [10][10] → 100 integers Ⓣ
- multidimensional array allocated Ⓕ