i. Imperative language: Assignment, Iteration, Order of execution is critical/ expose memory location <-> Functional: hide memory location

ii. OOP: a class can be a type (data and operations on that data bundled together)

iii. Dynamic Compilation: Compiling that takes place after running

iv. Just-in-Time Compilation: The program is stored in memory as byte code, but the code segment currently running is preparatively compiled to physical machine code in order to run faster

v. Subtype Polymorphism: 파라미터 타입 여러개 if one or more of its parameter types have subtypes (Animal cat1 = new Cat("Kitty");)

vi. Python Scope: Built-in – Global – Local (LGB)

vii. Labeled Namespace 영역 + 이름/ Primitive Namespace 눈에 보이지 않는 사소한 영역

viii. Static Scoping vs. Dynamic Scoping: environment 중요 – 동적으로 할당 받은 상황에 집중

ix. Static, Dynamic: 원래 Link 때 해야 하는데 Load 할 때 library 올리기

x. Delayed Linking – Dynamic linking 의 일부: Load 나 Run time 에 Linking 하는 방법 / Multiple programs can share a copy of library functions 여러 개 프로그램 / library functions can be updated independently of programs: all programs use repaired library code next time they run 한번에 수정/ Can avoid loading code that is never used (.dll/.so) 안 쓰는 코드 절약

xi. Load-time dynamic linking(.dll/.so): loader vs. Run-time dynamic linking(.dll): Running

xii. Profiling: Compiling 2 번

xiii. Activation of the function: the lifetime of one execution of a function from call to corresponding return → Activation record(Return address, Link to caller's activation record)

xiv. Activation-specific variable: each activation has its own binding of a variable to a memory location

xv. Static Allocation problem. Recursion & Multi-threading vs. Dynamic Allocation

2. Static allocation → Simple stack allocation – Stack Frame → Nesting link

i. Nesting Links: inner function to be able to find the address of the most recent activation for the outer function

ii. Displays: nesting links collected in a single static array

iii. Lambda lifting: Problem references replaced to new, hidden parameters

iv. Heap – 객체 생성 영역, 포인터로 연결 unordered runtime memory / Current heap link / Delayed Coalese(인접한 영역 합치기), Heap Compaction / Garbage Collection – Generational Collector, Incremental Collector

v. By value 값을 전달(copy-in, 초기값 O)	int z = plus(x,y);
vi. By result (copy-out, 별로 안씀, 초기값 X)	int z; plus(x,y,z);
vii. By value-result (copy-in/copy-out)	int z=1; plus(x,y,z);
viii. By reference 주소값 전달	Int z=1; plus(x,y,&z); → 함수 실행될 때 동시에 변하기
ix. By macro expansion(매크로, Capture)	#define MIN(X,Y) (~~)
x. By name (Macro Expansion without capture)	환경에 따라 이름이 딱 정해지게
xi. By need (꽤, on functional language) – Lazy evaluation	

- left associative, Right-associative 다시 확인
- Activation record 화살표 + Parameter passing
- Dynamic Scoping, Static Scoping
- EBNF, BNF
- Polymorphism
- Context Free Grammar