CLASE MARTES 05 SEPTIEMBRE ASTROINFORMÁTICA

- instrucciones en forma de algoritmo: este ordena las instrucciones para crear un flujo de transmisión de información

máquinas solo pueden leer lenguaje de máquina (o binario gigante)
python funciona con indentaciones, saltos de línea, con 2 puntos
C++ no ocupa indentaciones, ocupa punto y coma y llave

- cada lenguaje está asociado a paradigma de programación: 1ero: imperativo: decir al lgje instrucción x instrucción lo que quiero hacer

en python uso imperativo pq aún no uso orientación a objetos
declarativo: no implemento la solución, sino las características de mi problema y lo que quiero conseguir
orientación a objetos: ruptura en la forma que se programa por su versatilidad

- mi lenguaje puede ser interpretado o compilado
- tienen asociado un nivel de asociación, qué tan explícito debo programar dependiendo del lgje que estoy ocupando,

más alto : programo en términos de mi algoritmo, más abstracto, en bajo nivel explico todo.
(gauss creó la sumatoria cuando tenía 12 años :o)
alto nivel es la sintaxis de C
assembly: lenguaje ensamblador
toma instrucciones, las traduce a binario
Fortran77 demasiado antiguo x_x
alto nivel: C++
ORIENTACIÓN A OBJETO
Lisp (tiene otro paradigma), Julia
Python más cerca del lgje humano, más abstracto, posible menor eficiencia
python: tiene menos eficiencia :o porque con doble for keda la cagáaa
se necesitan librerías en C para q funcione x ejemplo numpy
Cython
pierde control

Assembly: memoria en el proceso , de la ram hacia el caché hay un buffer que se utiliza? a través de assembly
ventajas: sintaxis muy simple
drivers de telescopios son hechas only con assembly

telescopios modernos no incluyen bias
algoritmo alto nivel
ojito le falta otro bias restando al dark

orientación a objetos: gato vestido

niveles de abstracción
medio: C y C++, medio pa alto
porque tienen equilibrio entre abstracción y conexión con el hardware
totalmente utilizados en la programación de alto rendimiento
memoria dinámica y una estática: conocimiento sobre gestión de recursos
lgje de bajo nivel es más difícil de entender, lenguaje de alto nivel es más fácil de aprender

lenguaje compilado
código fuente es donde yo escribo mi lenguaje
múltiples pasos
2 grandes pasos: 1ero involucra traducir el código a lenguaje máquina, antes de eso debo verificar que el código entero sea consistente: léxico: no equivocarse al escribir (in y no on ekisde), sintáctico viene un ;, semántico: corresponda con el lgje
optimización: estrategias de programación que mejoran el rendimiento que ya se sabe que funcionan, ej: tengo un loop y dentro tengo una definición, esa definición debe estar afuera para evitar errores
OPTIMIZACIONES
cambia el ejecutable, no cambia el código, arregla todo automáticamente, debuggear un código compilado: no quiero optimización.

enlazador o linker: combina todos los archivos traducidos a lgje máquina y los combina con las librerías a usar, con eso genero ejecutable
error de instalar un software porque no encuentra una librería
linker tiene consideraciones que hay que tomar en cuenta

intérprete: python
lee instrucción x instrucción, loop no se puede hacer?
re-traducir mi código a código máquina cada vez que quiero compilar
me olvido de datos estáticos, puedo tener datos dinámicos, que cambien a tiempo real, puedo depurar o debugging en tiempo real, y eso no lo puedo hacer en compilado

C es lenguaje imperativo, hay que decirle el valor de retorno

cuando programo en C siempre tengo que definir una función como parte del código, por procedimiento

mkdir 2.1
en 2.1 vim asdf.c

gcc asdf.c

/n
./a.out
gcc asdf.c

brew install C
chmod +x le da permiso de ejecución

git pull para que se actualice solo

$ para llamar una variable en bash