



CS 35L

Discussion 1A

Week 2

Lisp and Python



Last Week

1. Absolute / Relative Path
 - a. The Unix File System
 - b. Command to navigate through the file system
2. Soft / Hard Links
 - a. Difference between soft and hard links - inode value
3. Shell Script
 - a. First line
 - b. Variables
 - c. Loops, if statements ...

Reminds

- Assignment 2 is due soon
 - Due: Oct 15 2022
 - **11:55** pm UCLA Time
 - Submission:
 - `gps-line.el`
 - `shuf.py`
 - `notes.txt` - answer questions and contain notes/comments
 - No `dribble` files are required
 - Grading Rubric: Piazza



Project

- Open-ended
 - Feel free to choose any ideas!
 - Great for your resume
- Key requirements
 - Some type of Client-Server Application
 - Front-end Tech
 - Back-end Tech
 - Applications should support:
 - Dynamic data, website updates based on what is sent back and forth to the server
 - Uploading: Client upload persistent data to server
 - Searching: Can search through server-side data
 - 3 more unique features, based on your project idea
- **Make sure you go over the descriptions on Class website:**

<https://web.cs.ucla.edu/classes/winter22/cs35L/project.html>

Project

- Find your group!
 - Sign-up sheet (before Oct 10 11:55pm):
<https://docs.google.com/spreadsheets/d/1gzeVRzZLyckVtzzpADIkOoajwr9vMltggegD-7Kb87E/edit?usp=sharing>
 - You can sign yourself up as a group
 - If you don't have enough ppl in your group, you can still sign up, and wait for other ppl to reach out
 - Or you can also go over the individual list to find you preferred teammates
 - If you are an individual
 - Reach out to groups you are interested in working with
 - Or just put yourself on the right-hand side, we will randomly allocate these students at the end
 - Please sign yourself up ASAP (Oct 10, Mon)
 - Team size: ≤ 5
- Initial Project Proposal and Schedule will due **Oct 15 and Oct 17** respectively

Project - Continue

- Project Proposal - Due: Oct 17 2022
 - High-level overview of your application.
 - What is the functionality of the app?
 - What is the User going to interact with?
 - What is the business logic?
 - Brief idea of how this implemented and how this will be feasible.
- Project Initial Schedule - Due: Oct 19 2022
 - Specify at least one (preferably two) intermediate milestones for the project.
 - Point of the milestones is to kind of push your group to move forward
 - The plan doesn't have to be perfect - it can be adjusted as the project evolves.

Contents

0. Programming Language

1. Lisp

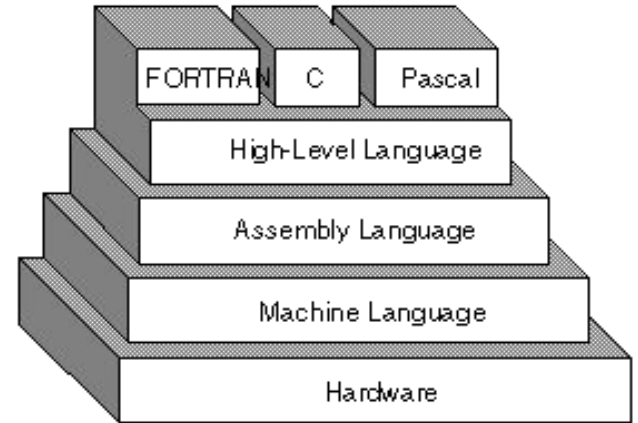
- a. Why Lisp
- b. Printing
- c. Arithmetic
- d. Variables and Data Type
- e. If Condition
- f. Loop
- g. Functions

2. Python

- a. Why Python
- b. Printing
- c. Variables and Data Type
- d. Conditions
- e. Loop
- f. Functions
- g. Module

0. Programming Language

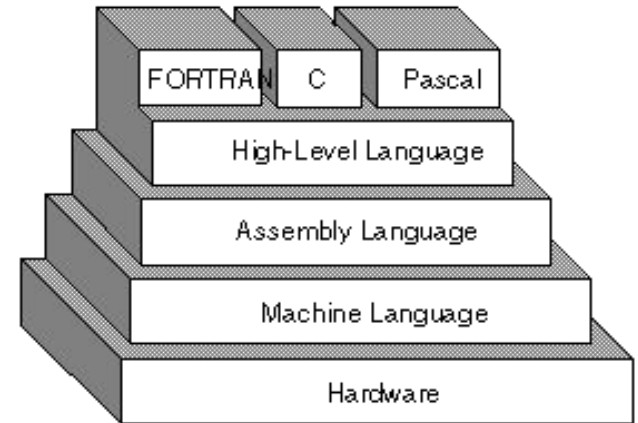
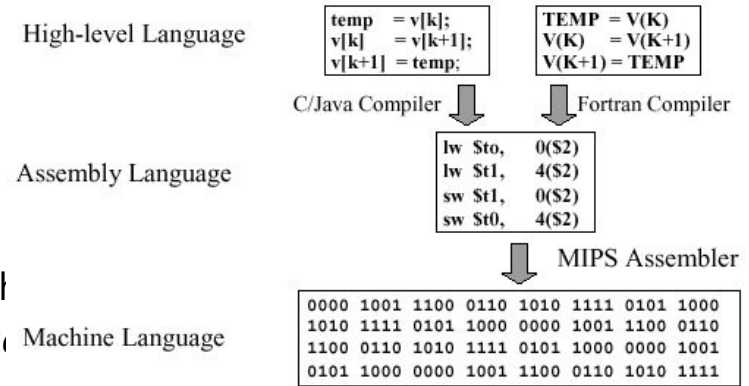
- A programming language is
 - a *formal language* comprising a *set* of *instructions* that produce various kinds of *outputs*
 - a *vocabulary* and set of *grammatical rules* for *instructing* a *computer* or computing *device* to *perform specific tasks*
- High-level programming language
 - For human understanding
 - A unique set of keywords and a special syntax => instructions
- Machine language
 - For machine understanding
 - Different CPU has its own unique machine language
- Assembly language
 - Lying between high-level and machine languages
 - Easier to program (allow name substitution)



0. Programming Language

- A programming language is
 - a *formal language* comprising a *set* of *instructions* to
 - a *vocabulary* and set of *grammatical rules* for instructions to *perform specific tasks*

- High-level programming language
 - For human understanding
 - A unique set of keywords and a special syntax => instructions
- Machine language
 - For machine understanding
 - Different CPU has its own unique machine language
- Assembly language
 - Lying between high-level and machine languages
 - Easier to program (allow name substitution)

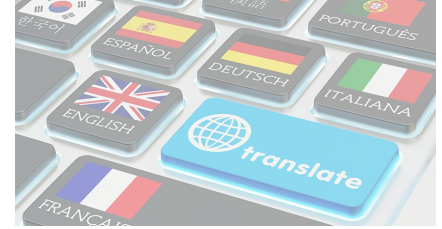


0. Programming Language

- High-level programming language => Machine language



0. Programming Language



- High-level programming language => Machine language
- Think about what human will do when reading instructions

- Method 1:
 - Read through **all the instructions**
 - Translate and record into our memories
 - Do some optimization in your head, e.g.: replan the path and order of the instructions
 - Follow the memories and do the instructions

Step to make instant Coffee



Put instant powder coffee to an empty cup.



Pour hot water into the cup



Stir with the spoon for mix



Happy with drink coffee

- Method 2:
 - Read the first instruction
 - Translate to sth we can understand
 - Do it
 - Read the second instruction
 - Translate to what we can understand
 - Do it
 -

0. Programming Language

- High-level programming language => Machine language

High-level Language

```
temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

```
TEMP = V(K)  
V(K) = V(K+1)  
V(K+1) = TEMP
```

C/Java Compiler

Fortran Compiler

Assembly Language

```
lw $to, 0($2)  
lw $t1, 4($2)  
sw $t1, 0($2)  
sw $t0, 4($2)
```

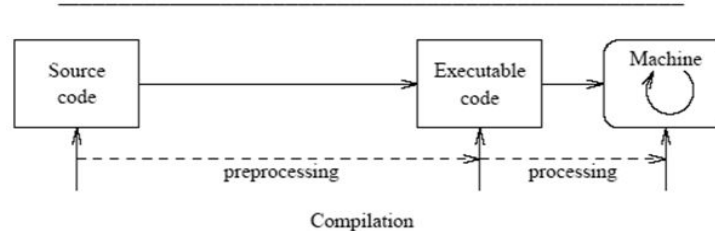
Machine Language

MIPS Assembler

```
0000 1001 1100 0110 1010 1111 0101 1000  
1010 1111 0101 1000 0000 1001 1100 0110  
1100 0110 1010 1111 0101 1000 0000 1001  
0101 1000 0000 1001 1100 0110 1010 1111
```

- **Compile** the program:
deal with the **whole program at once**

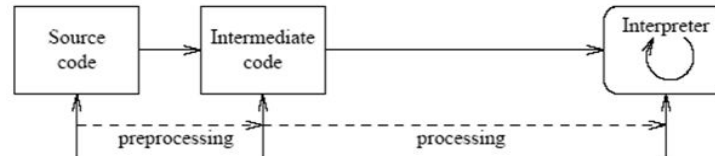
- Source code
- =(heavy)=>
- object code
- =>(linker)=>
- execute



Compilation

- **Interpret** the program:
translate programs **on the fly**

- Source code
- =(light)=>
- intermediate form
- =>
- execute



Interpretation

Contents

0. Programming Language

1. Lisp

- a. Why Lisp
- b. Printing
- c. Arithmetic
- d. Variables and Data Type
- e. If Condition
- f. Loop
- g. Functions

2. Python

- a. Why Python
- b. Printing
- c. Variables and Data Type
- d. Conditions
- e. Loop
- f. Functions
- g. Module

1. Lisp -- Why?

- Lisp was first developed in the late 1950s at the MIT for AI research.
- It's the second-oldest high-level programming language. Only Fortran is older, by one year.
- A wide family of languages and implementations
- Why Lisp?
 - Most of the Emacs functions are written in Lisp
 - Although Emacs Lisp is usually thought of in association only with Emacs, it is a full computer programming language
 - Extend Emacs (add/modify functions to Emacs)
 - Better understand fundamentals of programming
 - More importantly, to show you how you can teach yourself to go further

1. Lisp -- Printing

- Print something to stdout
 - `message`
 - String
 - Value of some variables

```
; Semicolon starts a comment

; printing a string
(message "hi")

; printing variable values
(message "Her age is: %d" 16)           ; %d is for number
(message "Her name is: %s" "Vicky")    ; %s is for string
(message "My list is: %S" (list 8 2 3)) ; %S is for any lisp expression
```

1. Lisp -- Variables and Data Types

- Data Types

- Integer, float

```
;; int to float
(float 3) ; 3.0
(truncate 3.3) ; 3
(floor 3.3) ; 3
(ceiling 3.3) ; 4
(round 3.4) ; 3
```

- String, number

```
;; string <-> number
(string-to-number "3")
(number-to-string 3)
```

- Variables

- Global variables -- `setq`

```
;; global variables, no declaration needed
(setq x 1) ; assign 1 to x
(setq a 3 b 2 c 7) ; multiple assignment
```

- Local variables -- `let`

- Define a local scope where variables works in

- `(let (var1 var2 ...) body)`
- `(let ((var1 val1) (var2 val2) (...) ...) body)`
- `body` - one or more lisp expressions; the body's last expression value is returned

```
;; local variables eg1
(let (a b)
  (setq a 3)
  (setq b 4)
  (+ a b)
) ; 7
```

```
;; local variables eg2
(let ((a 3) (b 4))
  (+ a b)
) ; 7
```


1. Lisp -- Arithmetic

- Basic calculations

- `+`, `-`, `*`
- `/`
 - Integer
 - Float
 - single digit decimal number such as `2.` needs a zero after the dot, like this: `2.0`. For example, `(/ 7 2.)` returns `3`, not `3.5`.
- Mod: `%`
- Power: `expt`

```
(+ 4 5 1) ; 10
(- 9 2) ; 7
(- 9 2 3) ; 4
(* 2 3) ; 6
(* 2 3 2) ; 12

;; integer part of quotient
(/ 7 2) ; 3

;; division
(/ 7 2.0) ; 3.5

;; mod, remainder
(% 7 4) ; 3

;; power; exponential
(expt 2 3) ; 8
```

1. Lisp -- Boolean

- True, False (No boolean datatype)

```
;; symbol nil is false  
;; nil is equivalent to empty list ()  
  
;; symbol t is true
```

- Boolean Functions

- and, or, not

```
; and, or  
(and t nil) ; nil  
(or t nil) ; t  
;; can take multiple args  
(and t nil t t t t) ; nil  
  
; not  
(not (= 3 4)) ; t  
(/= 3 4) ; t. "/=" is for comparing numbers only  
(not (equal 3 4)) ;t. General way to test inequality.
```

- Boolean Functions

- Compare: numbers, strings

```
;; compare numbers  
(< 3 5) ; less than ⇒ t  
(> 3 5) ; greater than ⇒ nil  
(<= 3 5) ; less or equal to ⇒ t  
(>= 3 5) ; greater or equal to ⇒ nil  
(= 3 3) ; equal ⇒ t  
(= 3 3.0) ; equal ⇒ t  
(/= 3 4) ; not equal ⇒ t  
  
;; compare string  
(equal "abc" "abc") ; t  
;; dedicated function for comparing string  
(string-equal "abc" "abc") ; t  
(string-equal "abc" "Abc") ; nil. Case matters
```

- Compare: equal v.s. = v.s. eq

- equal: test if two values have the same **data type** and **value**
- =: test if two **values** are the same
- eq: test if two args are the same **Lisp object**

```
(= 3 3.0) ; t  
(equal 3 3) ; t  
(equal 3 3.0) ; nil. Because data type doesn't match  
(equal "e" "e") ; t  
(eq "e" "e") ; nil. Because not the same object
```

1. Lisp -- If Conditions

- If Then Else

- `(if test body)`
- `(if test true_body false_body)`
- `(when test expr1 expr2 ...)`
 - `(if test (progn expr1 expr2 ...))`

```
(if (< 3 2) 7 8) ; 8

;; no false expression, return nil
(if (< 3 2) (message "yes") ) ; nil
(when (< 3 2) (message "yes") ) ;

;; if with blocks of expressions
(if something
  (progn ; true
    ...
  )
  (progn ; else
    ...
  )
)
```

- Block of Expressions

- Group several expressions together as one single expression
- `(progn ...)`
- Return the last expression in its body
- Similar to a block of code `{ ... }` in C-like languages

```
(progn (message "a") (message "b"))
;; is equivalent to
(message "a") (message "b")
;; return the last expression
(progn 3 4) ; 4
```

1. Lisp -- Loop

- Most basic loop
 - `(while test body)`
 - `body`: one or more lisp expressions

```
(setq x 0)

(while (< x 4)
  (message "number is %d" x)
  (setq x (1+ x)))
```

1. Lisp -- Functions

- Define a Function

- `(defun function_name (param1 param2 ...) "doc_string" body)`
- *body*: one or more lisp expressions

```
(defun myFunction ()  
  "Testing"  
  (message "Yay!"))  
  
;; return number*8 where number is a input parameter  
(defun multiply-by-eight (number)  
  (* 8 number))  
  
;; call a function  
(multiply-by-eight 6) ; = 48
```

Contents

0. Programming Language

1. Lisp

- a. Why Lisp
- b. Printing
- c. Arithmetic
- d. Variables and Data Type
- e. If Condition
- f. Loop
- g. Functions

2. Python

- a. Why Python
- b. Printing
- c. Variables and Data Type
- d. Conditions
- e. Loop
- f. Functions
- g. Module

2. Python -- Why?

- Python was conceived in the late 1980s by Guido van Rossum in the Netherlands.
- Python 2.0 was released in Oct 2000; Python 3.0 was released in Dec 2008.
- Why Python?
 - Popular!!!
 - Large and active community
 - Many powerful libraries and tools
 - Open-source resources
 - Design philosophy emphasizes code readability
 - with notable use of significant whitespace
 - Language constructs and object-oriented approach
 - aim to help programmers write clear, logical code
 - for both small and large-scale projects

2. Python -- Printing

- Print something
 - print() function

```
# starts a comment
# printing a string
print("hi")
# printing variable values
print("Her age is: ", 16)           # Her age is 16
age = 16      # define a variable with its value assigned as 16
print("Her age is: ", age)         # Her age is 16
# formatted output
print(f"Her age is: {age}")        # Her age is 16

import math
print(f"The value of pi is approximately {math.pi:.3f}." )
# The value of pi is approximately 3.142.
```


2. Python -- Variables and Data Types

- Data Types

- int, float

```
# int to float
int(4.5) # 4
float(4) # 4.0
```

- string, number

```
# string <-> number
str(457) # '457'
int('356') # 356
float('341.53') # 341.53
```

```
int('341.53')
# error: invalid literal for int() with base 10
```

- bool

```
# True
# False
```

- tuple: used to group data; immutable

```
year_born = ("Paris Hilton", 1981)
julia = ("Julia", "Roberts", 1967, "Duplicity", 2009, "Actress",
"Atlanta, Georgia")
print(julia[2]) # 1967
Julia[0] = "AnotherName"
# TypeError: 'tuple' object does not support item assignment
```

- list: store multiple items in a single variable

```
this_list = ["apple", "banana", "cherry"]
another_list = [1, 2, 3]
print(len(this_list)) # 3 (element number)
another_list[2] = 4 # access the list items
print(another_list) # [1, 2, 4]
```

- dic: store data values in *key:value* pairs

```
this_dic = {"apple": 3, "banana": 1, "cherry": 5}
print(this_dic["apple"]) # 3, access the list items
this_dic["orange"] = 3 # insert items
print(this_dic) # {"apple": 3, "banana": 1, "cherry": 5, "orange": 3}
```

- set: item collection, unordered and unindexed, no repeats

```
this_set = {1, 2, 3, 3, 1, 2}
print(this_set) # {1, 2, 3}
this_set.add(4)
This_set[2] # TypeError: 'set' object does not support indexing
```

2. Python -- If Conditions

- Boolean Functions
 - compare: `>`, `>=`, `<`, `<=`, `==`, `!=`
 - `and`, `or`, `not`
- `if`, `elif`, `else`
 - `if` (if the conditions are true, then execute the following)
 - `elif` (if the previous conditions were not true, then try this condition)
 - `else` (catches anything which isn't caught by the preceding conditions)
- Indentation and whitespace
 - Each block needs to be indented to be grouped together
 - Indicate scope (try to only use tabs or only use spaces; a mixture of both can mess up the grouping)

```
# example 1
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")

# example 2
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")

# short-hand if
if a > b: print("a is greater than b")

# short-hand if-else
a = 2
b = 330
print("A") if a > b else print("B")
```

2. Python -- Loop

- While Loop

- `while`
- `else` (optional, run a block of code once when the condition no longer is true)

- For Loop

- `for .. in ..`
 - List, dic, set, string, range(..)
- `range()` function
 - `range(5)` (values 0 to 4, 5 is not included)
 - `range(2, 30)` (start, end)
 - `range(2, 30, 3)` (start, end, increment)
- `else` (optional)

```
# example - while
i = 1
while i < 6:
    print(i)
    i += 1

# example - while-else
i = 1
while i < 6:
    print(i)
    i += 1
else:
    print("i is no longer less than 6")

# example - for in (string)
for x in "banana":
    print(x)

# example - for in (list)
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)

# example - for in (range)
for x in range(2, 30, 3):
    print(x)
else:
    print("Finally finished!")
```

2. Python -- Loop

- While Loop

- `while`
- `else` (optional, run a block of code once when the condition no longer is true)

- For Loop

- `for .. in ..`
 - List, dic, set, string, range(..)
- `range()` function
 - `range(5)` (values 0 to 4, 5 is not included)
 - `range(2, 30)` (start, end)
 - `range(2, 30, 3)` (start, end, increment)
- `else` (optional)

- Break and Continue

- `break` (**stop** the **loop** even if the while condition is true)
- `continue` (**stop** the **current iteration** and continue to the next)

```
# example - continue
# 1, 2, 4, 5, 6
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)

# example - break
# 1, 2, 3
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

2. Python -- Functions

- Create a function
 - Keyword: `def`
- Call a function
 - Function name + parenthesis

```
# example - def a function
def my_function():
    print("Hello from a function")

# example - call a function
my_function()

# example - def a function with arguments
def my_function2(fname):
    print(fname + " Refsnes")

# example - def a function with arguments
def my_function3(dirname, fname):
    print(dirname + fname + " Refsnes")

# example - call a function with arguments
my_function2("Emil")
my_function2("Tobias")
my_function2("Linus")

my_function3("dirname", "filename")
```

2. Python -- Modules

- Consider a module to be
 - a code library
 - a file containing a set of functions you want to include in your application
- Use a module
 - Keywords:
 - `import ...`
 - `from ... import ...`
- Virtual Environments
 - <https://docs.python.org/3/library/venv.html>
 - <https://piazza.com/class/l8b0hihfan26qq/post/156>

```
# save this code in a file named module.py
def greeting(name):
    print("Hello, " + name)

person1 = {
    "name": "Somename",
    "age": 18,
    "country": "Someplace"
}
```

```
# another file to run, let's say test.py
import module

module.greeting("Yuxing")

a = module.person1["age"]
print(a) # 18
```

```
# another way to implement test.py
from module import greeting, person1

greeting("Yuxing")

a = person1["age"]
print(a) # 18
```

Questions?

--- The End ---
