programming languages

Dr. Tuğberk Kocatekin
Istanbul Arel University

assemblers & assembly language

- When a programmer wanted to move the contents of *register 5* to *register 6*, they used to write **4056**.
- Later, they created a *mnemonic* system where it became;
  - MOV R5, R6
  - LD, ADDI, ST, HLT (Load, add, store, halt)
    - Basically, they added names to make them easier to write.
- These descriptive names are called *identifiers* or *program variables*.
- Assemblers **convert** these mnemonic names into machine language instructions.
- Therefore, instead of writing it in machine language, you write using these identifiers which makes it easier. And there is now a program which can convert that to machine language.
- The *mnemonic system* for representing programs is collectively called **assembly language**.

assembly language

- **collectively**. There is not one ASM.
- They are not great programming environments.
- Machine dependent.
  - Which means you need to write different assembly if you want your program to work on other computers.
- Still need to think as if you are writing machine language.
  - Designing a house thinking about nails, bricks and boards instead of concentrating on the architecture. It is easier to design the house if you can think in terms of windows, doors and rooms.
- We need **machine independent** code.

machine independent

- Programming language should not be dependent on the machine. The code we write should be run on every other machine.
- First languages to be developed for this are:
  - FORTRAN: FORmula TRANslator
  - COBOL: COmmon Business-Oriented Language
- Here, the primitives are not similar to machine language but *high-level*.
  - identifier = expression (a potential high-level primitive)

compilers & interpreters

- **Compilers** are *translators* which translates programs expressed with high-level primitives into machine language.
  - C, C++, Rust, Go, etc.
- **Interpreters** are an alternative to the previous. They are similar to *translators* but instead of producing a machine-language copy of the program to be run later, interpreter executes the program as they are translated.
  - Python, JavaScript, Ruby, Bash Scripting, etc.
- Some programming languages can both be compiled and/or run with interpreters.

advantages & disadvantages

- The resulting program can be executed number of times. Overhead of translation only occurs once. However, in *interpreted languages* every time you want to run the program, you need to translate it again and again. Therefore, they are less efficient.
- In interpreted languages, it is easy to code, test and change. Offers features like *dynamic typing*, they are *platform independent* and usually have smaller program size.
- It would make sense to use a *compiled* language for the intensive parts of an application, whereas interfaces (invoking the application) and less-intensive parts can be written in an interpreted language. They are also good for prototyping.

cross-platform software

- An application relies on the operating system to perform many tasks. It will use the window manager, file manager, etc.
- Therefore, programs are compiled for the operating system that they are going to work on.
- Cross-platform software is software that is independent from the operating system as well as the machine's hardware design.
- Interpreted languages are usually cross-platform.

programming paradigms

- Imperative (procedural)
  - Python, C, C++, ...
  - Traditional approach to programming. Find an algorithm to solve the problem and execute it as a sequence of commands.
- Object-oriented
  - Java, C++, C#, ..
- Functional
  - Haskell, LISP, Scheme, ...
- Declarative
  - Prolog, SQL, AMPL, ...
  - Contrast to imperative. Describe a problem rather than the algorithm. Not common.
  - Additional information: https://www.youtube.com/watch?v=Sg4U4r_AgJU

object-oriented programming (OOP)

- A software is viewed as a collection of units called: **objects**
- Objects can interact with each other
- Every object has certain *functions* called **methods**.
- Entire system can be constructed as a collection of objects, where every object knows how to respond.

OOP vs imperative

- Example: list of names
- This description of the objects is called a **class**.
- pokemon is a **class**, p1 is an **object**, name is an **attribute**, attack is a **method**.

```java
public class Pokemon {

        String name;

        void attack() {
                System.out.println('Attack');
        }
}

public class Main {
        public static void main(String []args) {
        Pokemon p1 = new Pokemon();
        p1.name = 'Pikachu';
        p1.attack();
        Pokemon p2 = new Pokemon();
        p2.name = 'Bulbasaur';
        p2.attack();
}
}
```

```python

name1 = 'pikachu'
name2 = 'bulbasaur'

def attack(pokemon):
        print(pokemon + 'Attack')

attack(name1)

```

OOP vs imperative

- Objects within OOP are also small imperative program units. OOP does not mean it is not imperative. Most OOP languages contains many features in the imperative languages.
- C++ (an oop language) is derived from C by adding object-oriented paradigm. Since Java and C# are derivatives of C++, they are similar too.

functional paradigm

- A program is viewed as an entity which accepts inputs and produces outputs. These are referred to as *functions*.
- Everything is done in functions:
  - `(Find_diff (Find_sum Old_balance Credits) (Find_sum Debits))`
  - imperative:

```
Total_credits = sum of all Credits
Temp_balance = Old_balance + Total_credits
Total_debits = sum of all Debits
Balance = Temp_balance - Total_debits
```

scripting languages

- A subset of imperative languages. Generally used for certain administrative tasks instead of doing complex programs. Such tasks are called **scripts**.
  - Perl, PHP, VBScript, Javascript, etc..

variables and data types

- Remember that we use descriptive names to reference a space in memory. That is called a **variable**.
- **Data type** gives us information as to how that variable is encoded.
  - int x = 4
  - Here, “int” is short for **integer** (data type) and **x** is the name of the variable.
- There are several data types which can also change with programming languages.
- We don’t need to *declare* the data type of a variable in every programming language. For example in Javascript we can write *var* and in Python we can just write the name of the variable.
  - How does the interpreter understand the data type?

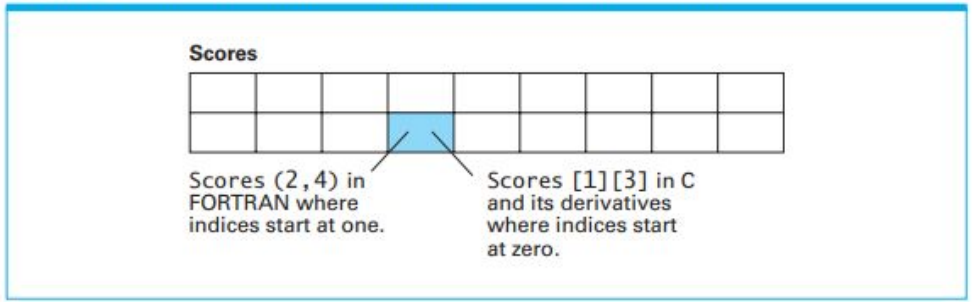
primitive data types & data structures

- Primitive data types are the data types that come built-in with the programming language.
  - int, char, boolean, float, etc.
- **Data Structures**
  - In addition to data type, variables are also associated with data structures.
  - A good example: **array.** There are many data structures: stack, queue, heap, linked list, etc.
  - Arrays are differently constructed in different languages. Therefore, they are declared differently too.
    - Python: myArray = []
    - C++: int myArray[4];

arrays

- You can also add more dimensions to an array to represent multiple dimensions.
  - int myArray[2][9];
  - myArray = [1,2,3]
  - You can also reference an element in this array by doing print(myArray[2]).
    - “2” here is called **index**.

Figure 6.5 A two-dimensional array with two rows and nine columns



control statements

- GOTO.
  - This was used a lot. Nowadays we use **if** instead of this. In Batch programming it is still used up to some extent. It makes hard to follow the program when it gets bigger.
- if, while, switch/case, for.

```
    goto 40
20  Evade()
    goto 70
40  if (KryptoniteLevel < LethalDose) then goto 60
    goto 20
60  RescueDamsel()
70  ...
```

```
if (KryptoniteLevel < LethalDose):
  RescueDamsel()
else:
  Evade()
```

functions

MORE IN THE NEXT LECTURE