Advanced #1: Programming Languages

SAMS SENIOR CS TRACK

Learning Goals

Understand the different types of **imperative** and **declarative** programming paradigms

Recognize other **variations** in programming language design have strengths and weaknesses

Identify the programming paradigms of **other languages**

Most of the information in this slide deck was developed based on Wikipedia's entry on Programming Paradigms. If you're interested, you can read more there!

Imperative vs. Declarative

Programming Paradigms

There are many different ways to approach programming. To start, we'll examine two main categories of programming languages.

Imperative programming languages are **state-based**. In programming, we think of state as the data that a program processes, the current values held in the computer's memory that can be accessed or changed. State-based programming revolves around the state and how to change it.

Declarative programming languages are **property-based**. These languages avoid changing the state of the program; instead, they declare what the result of a computation should look like. These properties are used to perform actual computation.

Imperative Programming

When programming imperatively, you usually focus on **how** the program should change the state over time. This is done with direct commands on variables/data structures that might update over time.

There are many different sub-categories of imperative programming languages. Two common ones are **procedural** and **object-oriented** programming, which are two different ways to organize the commands of programming.

Imperative – Procedural

Procedural programming revolves around describing different procedures (or functions) that can be performed on data, then calling those procedures on data directly.

Most of the programming we do in Python is procedural, though Python can be used in other programming paradigms as well. A common procedural language is C.

Imperative – Object-Oriented

Object-oriented programming still uses functions, but groups the functionality of the program into different **objects** to improve organization.

A classic theme in object-oriented programming is **inheritance**- build objects to inherit features and methods from each other, so that repeated work can be minimized.

In Python, we can create and use objects and inheritance. Read more here and here. Another common object-oriented language is Java.

Declarative Programming

When programming declaratively, you focus on **what** the result should look like, in terms of its properties. When the computation is expressed directly, the programming language itself can evaluate the code to determine what the result should be.

Many declarative languages aim to **avoid changing state**, and instead create new state when needed. This is done to avoid side effects, and to make it possible to mathematically prove when code works.

There are many different subfields of declarative programming. We'll look into two- **functional** and **logic** programming.

Declarative – Functional

Functional programming uses functions, like procedural programming. However, these functions do not track state; they derive the output directly based on the input.

This is often done by stating the returned value recursively, where the function calls itself on a different input. This is similar to a proof by induction- if we can derive f(x-1), we can derive f(x). Recursion will be next week's advanced topic.

We can write functional code in Python as long as we avoid changing state. This is easiest to do by using lambda, filter, map, and reduce; you can read more about these functions here. Another common functional programming language is Haskell.

Declarative - Logic

Logic programming sets up a series of logical facts and rules, and uses those facts and rules to derive new ideas. When the user sets the system a goal, the system attempts to achieve the goal by chaining together the facts and rules already known.

This is mainly used in mathematical settings, to derive new proofs. However, math can be applied in many fields of computer science, especially machine learning. This is also useful when attempting to find a solution that meets a certain set of constraints.

Python does not directly support logic programming, but there are external packages which can be imported in Python to perform these kinds of operations. A common logic programming language is Prolog.

Language Variations

Other Variations

Beyond Imperative and Declarative language styles, there are dozens of other models that programming languages can use to support different programming tasks. A list can be found here.

Beyond that, programming languages make many choices about how to represent syntax and process code. All of this variation means that there are hundreds of programming languages to choose from!

We'll look at a few different options that you might have noticed when comparing Python to a language you've learned before: **compiled** vs. **dynamic**, **weakly typed** vs. **strongly typed**, and **text** vs. **block**.

Compiled vs. Dynamic

When a computer runs a program, it needs to **parse** and **compile** the program before it can compute the result. This is the process the computer uses to understand how code should be executed at the machine level, where commands eventually turn into gate-level operations.

Some languages are **compiled-** the code must be fully compiled before it can be run. Compilation will often pre-perform some operations, to optimize how quickly the program runs when the user begins the process. Java is a common compiled language.

Other languages are **dynamic-** they re-compile the code every time the user runs it, and can add certain computations in as the program runs. These programs are often slower, as the computer can not pre-calculate results, but they also allow for more experimentation. Python is a common dynamic language.

Weakly vs. Strongly Typed

Most programming languages have a concept of **data types**. We've already gone over ints, floats, strings, and Booleans in this class.

Some languages have **weak typing**. Every variable has a type at runtime, but the type of the variable can change as the variable itself is updated with new values. This allows for more flexibility during code-writing. Python is a common weakly-typed language.

Some languages have **strong typing**. Variables must be assigned a type when they are defined, and the type may not change during runtime. Type changes are considered runtime errors in these languages. This prevents bugs caused by accidental type changes during computation. Java is a common strongly-typed programming language.

Text vs. Block

In all programming languages, the user must communicate the program to the computer in some way. But the modality used for communication can be different across different languages!

Most languages use **text**, like Python. The user must type out the text of the program, then the program parses the text into tokens and evaluates it. The syntax of the text varies across languages.

Some languages use **blocks** instead of text. These languages specify all of the different commands of the language into visual blocks, and the user can drag-and-drop them into an environment, arranging and nesting them as needed to achieve functionality. These blocks directly represent tokens, so syntax errors are rarer.

Bonus Task

Bonus Task

Choose a programming language that is **not** one of the following: Python, Java, HTML/CSS, Javascript. This should preferably be a language you have not used before. You can find a list of programming languages here: https://en.wikipedia.org/wiki/List_of_programming_languages

In a .txt file or as a comment in a .py file, answer the following questions:

- What language did you choose?
- Is your language mostly imperative, declarative, both, or neither?
- Which programming paradigm best describes your language?
- Describe a variation (like weak/strong typing, etc.) that makes this language interesting.

Submit your answer to the bonus1 assignment on Autolab by **noon on Friday 7/5**.