**Paradigms of Programming Notebook**

A programming paradigm is a fundamental style of programming: it is an approach to programming, defines the organising principle of a program, and it is a way to classify programming languages based on their features.

Programming paradigms are the result of people’s ideas about how computer programs should be constructed: models that serve as a "school of thoughts" for computer programming.

All programming languages can be categorised into programming paradigms.

There are generally two main programming paradigms: Imperative and Declarative, each are further divided into closely related sub-paradigms.

Languages that strictly lie within these paradigms are called "pure" in reality, very few languages are pure, most combine features of different paradigms.

A diagram of a diagram

Description automatically generated

**Imperative Programming:**

Computation is a series of steps (commands/instructions) executed in sequence:

programs are a sequence of commands to direct the computer to perform some computation, to control sequences, instructions can be run conditionally using if, and repeatedly using while/for, statements can change a program's state, principally through assignment operations or side effects.

Features: easy to implement, complex problems difficult to solve, parallel solutions cannot be implemented.

Disadvantages: code can become very long (Spaghetti Code), difficult to scale, less efficient and less productive.

Sub-paradigms: Procedural Programming, Object-Oriented Programming.

**Declarative Programming:**

Computation is done by defining what should be done (computed) and not how it should do (how that computation should be implemented).

Sounds like magic! Not really. It is left to the compiler to figure out the how. Declarative approaches rely on preconfigured functionality in the language to execute a task.

without explicit instructions on what steps to take to implement that task. It all predefined! In practice, this approach requires a domain-specific language (DSL) for expressing what tasks a programmer wants and shields them from the detail that defines how that task is implemented.

Features: short and efficient code, programs have no state change, and no side effects.

Disadvantages: sometimes hard to understand (code based on unfamiliar conceptual model), and despite potential for reusability, customising code may need significant rewrites.

Sub-paradigms: Functional Programming, Logic Programming.

**Common Concepts:**

Unifying language concepts: Data types (both built-in and user-defined), Expressions (e.g., arithmetic, Boolean, strings) , Functions/procedures/methods

Fundamentally, Computers can perform calculations and remember results.

A computer program is a set of instructions that tell a computer what to do: everything a computer does is controlled by a program.

**Syntax:**

The arrangement of words and phrases to create well-formed sentences in a language, the structure of statements in a computer programming language.

**Semantics:**

Semantics is concerned with the meaning of (programming) languages. i.e., the logic of the code, usually much more difficult to define than syntax.

**IDE:**

Integrated Development Environment.

An IDE is a software application for writing code that also provides comprehensive facilities for software development.

IDEs normally consists of at least: a source code editor, build automation tools, and a debugger.

**Python:**

Python is a multi-paradigm (hybrid) programming language: – Primarily Object-Oriented, but also supports many elements of:

* Procedural Programming: code is broken into functions, commands within these functions are executed sequentially, and can have side-effects (Side-effects happen if a function modifies some state variable value(s) outside that function).
* Functional Programming (FP): Python functions can return only one value,  
  these functions can deign to disallows side effects, and o other functional programming features also supported.

Interpreted means that Python code is converted to machine code and executed by another program (the Python interpreter) one statement at a time at run-time.

﻿﻿For Compiled code, the entire source code is compiled into machine code on one machine and then executed altogether on a target machine. Whether the code is interpreted or compiled is not a property of the language but a property of the implementation.

**Commands:**

print(‘Hello World!’)

print is the function and what is between parentheses is called function argument.

print(‘My name is /”Arad Soutehkeshan/” and I am 17’)

output: My name is “Arad Soutehkeshan” and I am 17

print(‘’’Hello,

How are you?’’’)

Output: Hello,

How are you?

**Variable:**

A container of data (a value) that can change while the program runs; this container is a place in computer memory which has been especially allocated to the variable (allocation is temporary, only while a program runs); the place in memory is allocated when the variable is created/defined.

A variable name (identifier) is a reference to the memory location allocated.

It’s better to write constant variable names in upper case.

**Comments:**

Comments are text notes about the code and are used mainly for providing explanatory notes about the code; this helps other programmers understand your code and its purpose, and also helps you remember your own code if you've left it for a while; for keeping versions of parts of the code that are not working properly but you want to keep (# or ‘’’ ‘’’).

‘’’ ‘’’ will have memory so #s are better.

**Data types:**

Data Type: A category characterising a single or set of data values and constrains the operations that can be performed on that data (e.g., int, float, str, bool, UDT, complex numbers).

When there is a division sign the answer is always float and should be written as 218.0.

**Statement, Expressions and Operators:**

﻿A statement is an instruction/command the Python interpreter can execute (e.g., x=1).

An expression calculates a value or evaluates some logic:  it is any combination of values, variables and operators that produce a new value when evaluated (executed or run);spaces in an expression are not signific (e.g., x/10).

Operators are special symbols in Python that carry out arithmetic (e.g., +, -, /, etc.)  
or logical computation.

The value that the operator operates on is called the operand (e.g., x+y).**A diagram with arrows and symbols

Description automatically generated with medium confidence**

**Control Structures:**

Control structures allow us to alter this sequential program flow. Also called decision structures or conditional statements. Allow different blocks of code to be executed based on some condition.

**A close-up of a diagram

Description automatically generated**

***Flow Control:***

**If Statement:**

The Python if statement is used to implement conditionals (decisions).

• Syntax: if <condition>: <body>

* <body> is a statement or a block of statements indented under the if heading.
* <body> must be indented.

• Semantics

First, <condition> is evaluated. If it is True, <body> is executed, then control passes to the next statement after the conditional. If False, <body> is skipped, and control passes to next statement after conditional.

The if statement is a one-way conditional (decision) statement. It has only one branch. If condition is true, control flow passes to the true block (branch) and its statements are executed sequentially; otherwise, control passes to the next statement in program that is directly after the conditional statement.

Parentheses are only needed for compound expressions or to improve readability.

**Blocks:**

Python uses indentation (whitespaces at the start of a line) to denote blocks of code.

A colon: at the end of a line indicates that the next line of code is the beginning of a block.

Lines within the code block are indented at the same level.

Remove indentation to denote end of a code block.

Blocks are needed if you want parts of your code to run only when certain conditions are met.

The if statement allows a particular block of code to run if its condition evaluates to True.

What if you need to run a different block of code if the condition evaluates to False or if you need more than two alternatives.

A selection statement provides the means of choosing between two or more paths of execution.

Two general categories: Two-way conditionals (if-else statements), Multiple-way conditionals (if-elif statements)

**Two-way Conditionals:**

If statements are used as guarded actions: only execute a block if a condition evaluates to True.  
Another programming pattern is to do one thing or another. It has two branches or paths.

– if condition is true, control flow passes to the true block (branch); otherwise, control passes to the false block; when either block completes, control passes to the next statement in program that is directly after the conditional statement.

**If-else statements:**

In Python, a two-way decision can be implemented by attaching an else clause onto an if statement. This is called an if-else statement.

• Syntax: if<condition>: <body>

else: <body>

• Semantics:  
Python first evaluates the condition. If condition is True, the then clause (true block) is executed. If condition False, the else clause (false block) is executed. When either body completes, control passes to the line after conditional.

**Nested Conditionals:**

The then and else blocks of an if-statement can contain other conditionals too. This is called nested conditionals or nested if-statements. A nested if statement means an if-statement inside another if-statement. This is used in situations where you want to check for other condition(s) after a condition evaluates to true.

Syntax:

if <condition>: <body>

if <condition1>:

<body>

else: <body>

else:   
 if <condition2>:

<body>

else:

<body>

**Multi-way Conditionals:**

Allow selection of one of any number of statements or statement blocks. Useful if the program needs more than two alternative decisions.

• Syntax: if <condition1>: <body>

elif <condition2>: <body>

elif <condition3>: <body>

else: <default body>

This form of conditional statement sets any number of mutually exclusive code blocks.

Exactly one branch will be executed.

**Ternary Operators:**

Ternary operators allow the evaluation of a condition in a single line by replacing the multiline if-else statement, making the code compact.

Alternative syntax: <on\_True> if <condition> else <on\_False>

Example:

#get smallest of two values

smallest = a if a < b else b

if x!=4: x+=y; print(x); print(y)

Semi colons are used to delimit statements if multiple statements are put on the same line.

If <condition> is True, all three statements are executed, otherwise, that block is skipped.

The following is also legal in python: x=4; y=5; z=y\*x; print(z)

The pass keyword indicates that nothing happens when it is executed. sometimes useful, e.g., during development

**Import:**

Import allows you to use code defined elsewhere (either from a standard module, or your own). A module is simply a file containing Python code.

**Boolean:**

Booleans are expressions that always evaluate to either True or False. They produce a value of type bool (Boolean value).  
Example: a > b *#* true if a is greater than b; false otherwise.

Named after the English mathematician George Boole.

These expressions are formed with Relational and/or Logical operators:

Relational (comparison) operators:

* + Determines whether a specific relationship exists between two values.
  + Example: greater than (>) a > b

Logical operators:

* + Used to create complex and compound Boolean expressions.
  + Example: a > b and b < c

Syntax:  
<expr> <rlop> <expr>

where <rlop> is short for relational or logical operator; <expr> each operand can be a value, variable or another expression.

And and or operators:  
– binary operators (i.e., have two operands),  
– connect two Boolean expressions into a compound Boolean expression:

Example: height >= 180 and age < 18

Not operator:

– unary operator (i.e., has one operand), inverts the truth of its Boolean operand (i.e., inverts its value).

Example:

x = True

print(not x)

output = False