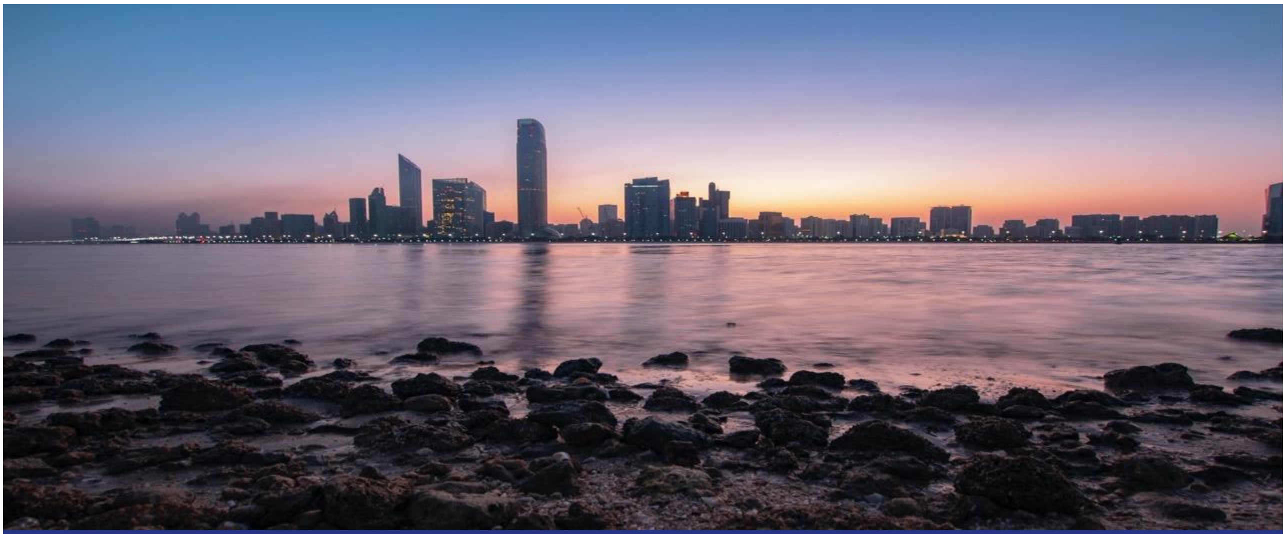


Fundamentals of Codification

29th – 30th October 2023

1

Course Agenda

01 Overview of Programming Languages

- Definition of a programming language
- High-level and low-level languages
- Programming paradigms
- Types of programming languages
- Execution of programming languages

02 Popular programming languages

- Cobol
- Python
- Java
- JavaScript
- C#
- Rust
- Go
- Swift

03 Introduction to Python

- Overview of Python and its applications
- Dynamic typing
- Basic operators
- Basic data types
- Whitespace
- Comments
- Sequence types
- Positive and negative indices
- Slicing: Return copy of a subset
- Operations on lists only
- Dictionaries: A mapping type
- Creating and accessing dictionaries
- Removing dictionary entries
- Useful accessor methods

04 Application of Python in Banking and FIs

- Handling data using Python libraries (NumPy and Pandas)
- Data manipulation and cleaning
- Data visualization using Matplotlib and Seaborn
- Exploratory data analysis – using Bank loan default

2

Overview of Programming Languages

Learning Outcome 1

3

Definition of a Programming Language

A programming language is a formal system for instructing computers, using rules and syntax, to perform specific tasks. It acts as a bridge between human-readable code and machine-executable instructions, enabling the creation of software applications, from simple scripts to complex programs.

4

High-level & Low-level Languages

High-Level Language:	Low-Level Language:
• Designed for user-friendliness and abstraction from computer hardware. • Uses natural language elements and is closer to human-readable code. • Focuses on problem-solving and algorithm implementation. • Offers built-in functions and libraries for efficiency. • Examples: Python, Java, C++, JavaScript.	• Closer to hardware, providing direct hardware control. • Involves writing architecture-specific code. • Less human-readable, requires deep hardware understanding. • Used for tasks needing precise hardware control. • Example: Assembly language.

5

Programming Paradigms

Programming paradigms are fundamental approaches to designing and structuring computer programs. They define the principles and concepts that guide software development. Various programming paradigms exist, each offering its own set of rules and ways of organizing code. Here are some common programming paradigms:

Imperative Programming: This approach involves specifying the steps needed to achieve a specific goal. It uses statements that change the program's state through assignments, loops, and conditionals. Languages like C and Fortran follow this paradigm.

Functional Programming: Functional programming treats computation as the evaluation of mathematical functions. It emphasizes using pure functions (those with consistent outputs for the same inputs) and avoids mutable data. Languages like Haskell and Lisp are known for functional programming.

6

Programming Paradigms

- **Object-Oriented Programming (OOP):** OOP revolves around objects, instances of classes. Objects encapsulate data and behavior, and interactions between objects drive the program's logic. Java, C++, and Python support OOP.

- **Procedural Programming**: Similar to imperative programming, procedural programming focuses on functions or procedures. It encourages dividing the program into reusable functions and avoiding global variables. C can be used in a procedural programming style.

- **Declarative Programming**: Declarative programming describes what should be done, rather than specifying step-by-step procedures. SQL for databases and HTML for web pages are examples of declarative languages.

Programming Paradigms

- **Logic Programming:** Logic programming is based on formal logic. Programs are written as sets of logical rules, and computation involves proving or disproving these rules. Prolog is a well-known logic programming language.

- **Event-Driven Programming:** In this paradigm, programs respond to events, such as user actions or system messages. It often uses callbacks or event handlers to trigger actions in response to events, common in GUI and web development.

- **Concurrent and Parallel Programming:** This approach deals with managing multiple tasks running concurrently or in parallel for improved performance. Languages like Go and Erlang provide features for concurrent and parallel programming.

- **Aspect-Oriented Programming (AOP):** AOP separates cross-cutting concerns (e.g., logging, security) from the main program logic using aspects that can be woven into the code at specific points.

Types of Programming Languages

Procedural Programming Languages:

Characteristics	Examples	Common Uses
• Procedural languages focus on procedures or routines. They use a step-by-step approach to solve problems, with an emphasis on functions and subroutines.	• C, Pascal, Fortran	• System programming, scientific computing, embedded systems.

9

Types of Programming Languages

Object-Oriented Programming (OOP) Languages:

Characteristics	Examples	Common Uses
• OOP languages use objects as the building blocks of programs. They emphasize encapsulation, inheritance, and polymorphism.	• Java • C++ • Python	• General-purpose programming, web development, game development.

10

Types of Programming Languages

Functional Programming Languages:

Characteristics	Examples	Common Uses
• Functional languages treat computation as the evaluation of mathematical functions. They emphasize immutability and pure functions.	• Haskell • Lisp • Scala	• Mathematical modeling, artificial intelligence, data analysis.

11

Types of Programming Languages

Scripting Languages:

Characteristics	Examples	Common Uses
• Scripting languages are often interpreted and dynamically typed. They are used for automating tasks, scripting, and rapid development.	• Python • Ruby • JavaScript	• Web development, automation, data analysis, system administration.

12

Types of Programming Languages

Markup Languages:

Characteristics	Examples	Common Uses
• Markup languages are used to describe and present structured data. They use tags to define elements and their attributes.	• HTML • XML • LaTeX	• Web page creation, document formatting, data interchange.

13

Types of Programming Languages

Domain-Specific Languages (DSLs):

Characteristics	Examples	Common Uses
• DSLs are tailored for specific domains or industries. They provide specialized syntax and functionality for particular tasks.	• SQL (for databases) • VHDL (for hardware design) • MATLAB (for mathematical modeling)	• Database querying, hardware description, scientific simulations.

14

Types of Programming Languages

Low-Level Languages:

Characteristics	Examples	Common Uses
• Low-level languages provide direct control over hardware and memory. They are less abstract and closer to machine code.	• Assembly languages • Machine code	• Operating system development, embedded systems, device drivers.

Types of Programming Languages

Concurrency and Parallelism Languages:

Characteristics	Examples	Common Uses
• These languages support concurrent and parallel programming, enabling multiple tasks to run simultaneously.	• Go • Erlang • MPI (Message Passing Interface)	• Distributed systems, real-time applications, high-performance computing.

Types of Programming Languages

Visual Programming Languages:

Characteristics	Examples	Common Uses
• Visual languages use graphical elements instead of text. Users build programs by connecting visual components.	• Scratch • Blockly • LabVIEW	• Education, prototyping, graphical user interface design.

17

Execution of Programming Languages

Programming languages can be executed using various methods, including compilers, interpreters, and JVM-based execution. Each method operates differently:

18

Execution of Programming Languages

Compiler:

Overview	Process	Characteristics
• A compiler translates entire high-level source code into machine code or an intermediate form in one go, creating an executable file.	• The programmer writes code, the compiler checks it for errors, and if successful, generates an executable.	• Compilation is a one-time process, resulting in fast execution. Examples include C and C++.

Execution of Programming Languages

Interpreter:

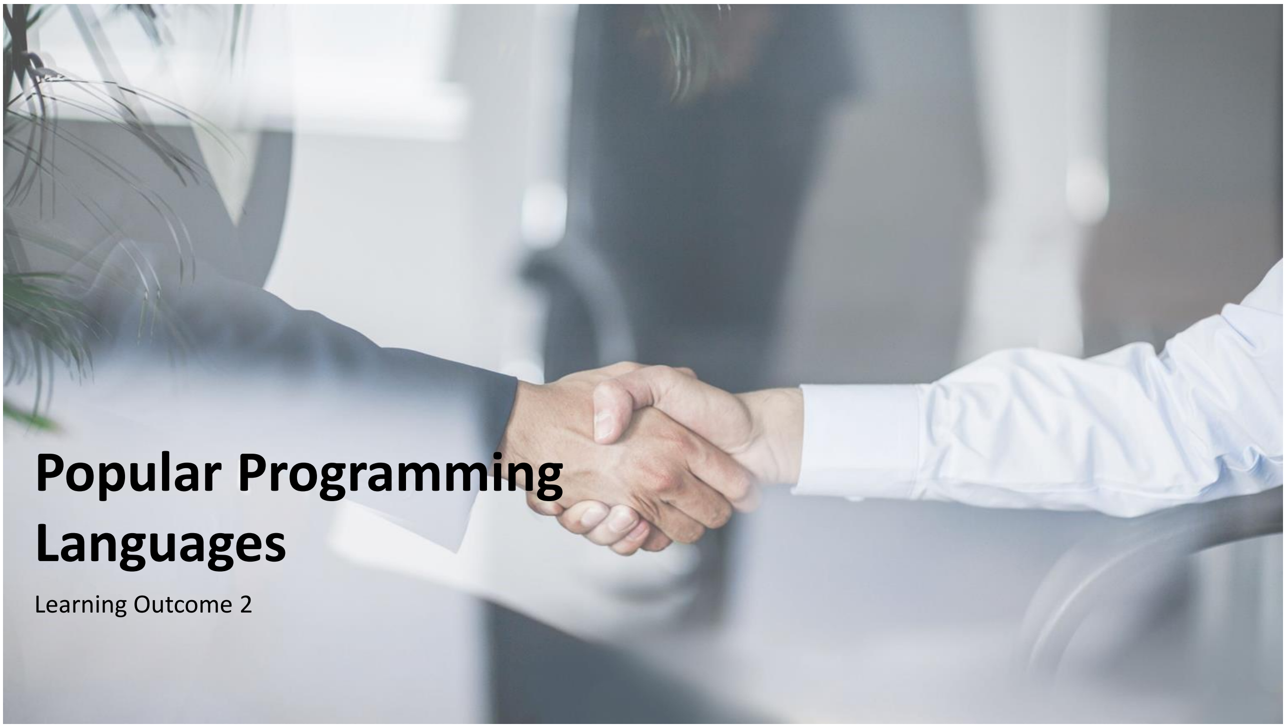
Overview	Process	Characteristics
• An interpreter executes source code line by line in real-time, without creating a separate executable.	• Code is read and executed line by line, errors are reported immediately.	• Easy debugging, but execution can be slower. Languages like Python and Ruby use interpreters.

Execution of Programming Languages

JVM (Java Virtual Machine)-Based Execution:

Overview	Process	Characteristics
• JVM-based languages (e.g., Java) compile source code into platform-independent bytecode, executed by the JVM.	• Code is compiled to bytecode, executed by the JVM, and translated to machine code for the host system.	• Offers platform independence and runtime optimization. Java and Kotlin use this method.

21



22

Popular Programming Languages

COBOL (Common Business-Oriented Language):

Key Features
- Designed for business applications, readable by non-programmers, strong support for file handling and data manipulation.

When to Use
- COBOL is still used in legacy systems, particularly in the finance, banking, and government sectors.

Pros
- Readable, stable, and well-suited for large-scale data processing.

Cons
- Older syntax, limited support for modern programming paradigms.

Year of Establishment
- 1959

Popular Programming Languages

Python:

Key Features
- Easy-to-read syntax, vast standard library, strong community support, versatile for web development, data science, automation, and more.

When to Use
- Python is suitable for a wide range of applications, including web development (Django, Flask), data analysis (pandas, NumPy), and scripting.

Pros
- Easy to learn, versatile, strong community support.

Cons
- Slower execution speed compared to some languages.

Year of Establishment
- 1991

Popular Programming Languages

Java:

Key Features
- Platform independence, strong typing, extensive libraries (Java Standard Library), used in enterprise applications, Android app development.

When to Use
- Java is ideal for developing cross-platform desktop and web applications, as well as Android mobile apps.

Pros
- Portability, scalability, and robustness.

Cons
- Verbosity, slower execution than some natively compiled languages.

Year of Establishment
- 1995

Popular Programming Languages

JavaScript:

Key Features
- Client-side scripting language, essential for web development, supports asynchronous programming, used with HTML and CSS.

When to Use
- JavaScript is a must for front-end web development and can also be used for server-side scripting (Node.js).

Pros
- Ubiquitous, versatile, excellent for user interfaces.

Cons
- Can be challenging to debug, browser compatibility issues.

Year of Establishment
- 1995

Popular Programming Languages

C# (C-Sharp):

Key Features
- Developed by Microsoft, used for Windows applications, strong in desktop and game development (Unity), .NET framework.

When to Use
- C# is suitable for Windows desktop applications, game development, and enterprise software within the .NET ecosystem.

Pros
- Versatile, strong tooling and IDE support.

Cons
- Platform-dependent, primarily used in the Microsoft ecosystem.

Year of Establishment
- 2000

Popular Programming Languages

Rust:

Key Features
- Focuses on memory safety, performance, and zero-cost abstractions, prevents common programming errors, system-level language.

When to Use
- Rust is a good choice for system programming, embedded systems, and when safety and performance are critical.

Pros
- Memory safety, performance, modern features.

Cons
- Steeper learning curve, smaller ecosystem compared to older languages.

Year of Establishment
- 2010

Popular Programming Languages

Go (Golang):

Key Features

- Created by Google, simplicity, and efficiency, built-in concurrency support (goroutines), used for web servers, microservices, and cloud computing.

When to Use

- Go is well-suited for building scalable and efficient web services and server-side applications.

Pros

- Speed, simplicity, built-in concurrency.

Cons

- Smaller standard library compared to some languages.

Year of Establishment

- 2009

29

Popular Programming Languages

Swift:

Key Features

- Developed by Apple, designed for iOS, macOS, watchOS, and tvOS app development, modern syntax, memory safety.

When to Use

- Swift is the primary language for Apple ecosystem app development, including mobile and desktop applications.

Pros

- Safety, performance, modern language features.

Cons

- Limited to Apple platforms, less cross-platform support.

Year of Establishment

- 2014

30

Introduction to Python

Learning Outcome 3

31

Why Python?

Object Oriented:
- With functional constructs: map, generators, list comprehension

NLP Processing: Symbolic:
- Python has built-in datatypes for strings, lists, and more

NLP Processing: Statistical:
- Python has strong numeric processing capabilities: matrix operations, etc.
- Suitable for probability and machine learning code.

NLTK: Natural Language Tool Kit
- Widely used for teaching NLP
- Implemented as a set of Python modules
- Provides adequate libraries for many NLP building blocks

Google "NLTK" for more info, code, data sets, book.

Powerful but unobtrusive object system
- Every value is an object
- Classes guide but do not dominate object construction

Powerful collection and iteration abstractions
- Dynamic typing makes generics easy

32

Why Python?

Interpreted language
- It works with an evaluator for language expressions

Dynamically typed
- Variables do not have a predefined type

Rich, built-in collection types
- Lists
- Tuples
- Dictionaries (maps)
- Sets

Concise

33

Language Features

Indentation instead of braces

Newline separates statements

Several sequence types:
- Strings '…': made of characters, immutable
- Lists […]: made of anything, mutable
- Tuples (…) : made of anything, immutable

Powerful subscripting (slicing)

Functions are independent entities (not all functions are methods)

Exceptions

34

Dynamic Typing

- Variables come into existence when first assigned to
- A variable can refer to an object of any type
- All types are (almost) treated the same way
- Main drawback: type errors are only caught at runtime

Playing with Python (1)

```
>>> 2+3
5
>>> 2/3
0
>>> 2.0/3
0.66666666666666663
>>> x=4.5
>>> int(x)
4
```

Playing with Python (2)

```
>>> x='abc'
>>> x[0]
'a'
>>> x[1:3]
'bc'
>>> x[:2]
'ab'
>>> x[1]='d'
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    x[1]='d'
TypeError: 'str' object does not support item assignment
```

37

Playing with Python (3)

```
>>> x=['a','b','c']
>>> x[1]
'b'
>>> x[1:]
['b', 'c']
>>> x[1]='d'
>>> x
['a', 'd', 'c']
```

38

Basic Operators (1)

- **Indentation matters to the meaning of the code:**
  - Block structure indicated by indentation
- **The first assignment to a variable creates it.**
  - Variable types don't need to be declared.
  - Python figures out the variable types on its own.
- **Assignment uses = and comparison uses = =.**

Basic Operators (2)

- **For numbers *+ - * / %* are as expected.**
  - Special use of + for string concatenation.
  - Special use of % for string formatting (as with print f in C)
- **Logical operators are words (and, or, not) *not* symbols**
- **Simple printing can be done with print.**

Basic Data Types

- Integers (default for numbers)

  z = 5 / 2   # Answer is 2, integer division.

- Floats

  x = 3.456

- Strings
  - Can use "" or '' to specify.

    "abc" 'abc' (Same thing.)

  - Unmatched can occur within the string.

    "matt's"

  - Use triple double-quotes for multi-line strings or strings than contain both ' and " inside of them:

    """a'b"c"""

Exercise 1: To do List (with a budget)

- Description
  - Create a program to manage tasks and track expenses within a specified budget.

Whitespace

Whitespace is meaningful in Python: especially indentation and placement of newlines.

- Use a newline to end a line of code.
  - Use \ when must go to next line prematurely.
- No braces { } to mark blocks of code in Python… Use consistent indentation instead.
  - The first line with less indentation is outside of the block.
  - The first line with more indentation starts a nested block
- Often a colon appears at the start of a new block. (E.g. for function and class definitions.)

Comments

Start comments with # – the rest of line is ignored.

Can include a "documentation string" as the first line of any new function or class that you define.

The development environment, debugger, and other tools use it: it's good style to include one

Sequence Types

- **Tuple**
  - A simple immutable ordered sequence of items
  - Items can be of mixed types, including collection types
- **Strings**
  - Immutable
  - Conceptually very much like a tuple
  - (8-bit characters. Unicode strings use 2-byte characters.)
- **List**
  - Mutable ordered sequence of items of mixed types

Sequence Types

- All three sequence types (tuples, strings, and lists) share much of the same syntax and functionality.
- **Key difference:**
  - Tuples and strings are immutable
  - Lists are mutable
- The operations shown in this section can be applied to all sequence types
  - most examples will just show the operation performed on one

Sequence Types

- **Tuples are defined using parentheses (and commas).**

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
```

- **Lists are defined using square brackets (and commas).**

```
>>> li = ["abc", 34, 4.34, 23]
```

- **Strings are defined using quotes (", ', or """).**

```
>>> st = "Hello World"
>>> st = 'Hello World'
>>> st = """This is a multi-line
string that uses triple quotes."""
```


Sequence Types

- We can access individual members of a tuple, list, or string using square bracket "array" notation.
- Note that all are <u>0 based</u>…

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]     # Second item in the tuple.
 'abc'
>>> li = ["abc", 34, 4.34, 23]
>>> li[1]      # Second item in the list.
 34
>>> st = "Hello World"
>>> st[1]   # Second character in string.
 'e'
```

Positive and Negative Indicies

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Positive index: count from the left, starting with 0.

```
>>> t[1]
'abc'
```

Negative lookup: count from right, starting with –1.

```
>>> t[-3]
4.56
```


Slicing: Return Copy of a Subset

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
```

Return a *copy* of the container with a subset of the original members. Start copying at the first index, and stop copying *before* the second index.

```
>>> t[1:4]
('abc', 4.56, (2,3))
```

You can also use negative indices when slicing.

```
>>> t[1:-1]
('abc', 4.56, (2,3))
```

The ‘in’ Operator

- Boolean test whether a value is inside a collection often called a container in Python:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- For strings, tests for substrings

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

- Be careful: the in keyword is also used in the syntax of for loops and list comprehensions.

The + Operator

- The + operator produces a new tuple, list, or string whose value is the concatenation of its arguments.

```
>>> (1, 2, 3) + (4, 5, 6)
 (1, 2, 3, 4, 5, 6)
>>> [1, 2, 3] + [4, 5, 6]
 [1, 2, 3, 4, 5, 6]
>>> "Hello" + " " + "World"
 'Hello World'
```

Operations on Lists Only (1)

```
>>> li = [1, 11, 3, 4, 5]
>>> li.append('a') # Note the method syntax
>>> li
[1, 11, 3, 4, 5, 'a']
>>> li.insert(2, 'i')
>>>li
[1, 11, 'i', 3, 4, 5, 'a']
```


The Extend Method vs. The + Operator

- + creates a fresh list (with a new memory reference)
- extend operates on list li in place.

```
>>> li.extend([9, 8, 7])
>>> li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7]
```

Confusing:

- extend takes a list as an argument.
- append takes a singleton as an argument.

```
>>> li.append([10, 11, 12])
>>> li
[1, 2, 'i', 3, 4, 5, 'a', 9, 8, 7, [10, 11, 12]]
```

Exercise 2: Weight Converter or BMI

- Description
  - Build a tool that can convert weight units or calculate BMI based on user input and classify individuals accordingly.

55

Operations on Lists Only (2)

```
>>> li = ['a', 'b', 'c', 'b']
>>> li.index('b')     # index of first occurrence
1
>>> li.count('b')     # number of occurrences
2
>>> li.remove('b')    # remove first occurrence
>>> li
['a', 'c', 'b']
```

56

Operations on Lists Only (3)

```
>>> li = [5, 2, 6, 8]
>>> li.reverse()    # reverse the list in place
>>> li
[8, 6, 2, 5]
>>> li.sort()       # sort the list in place
>>> li
[2, 5, 6, 8]
>>> li.sort(some_function)
   # sort in place using user-defined comparison
```

57

Dictionaries: A Mapping Type

- Dictionaries store a mapping between a set of keys and a set of values.
  - Keys can be any immutable type.
  - Values can be any type
  - Values and keys can be of different types in a single dictionary
- You can
  - define
  - modify
  - view
  - lookup
  - delete

  the key-value pairs in the dictionary.

58

Creating and Accessing Dictionaries

```
>>> d = {'user':'bozo', 'pswd':1234}
>>> d['user']
'bozo'
>>> d['pswd']
1234
>>> d['bozo']
Traceback (innermost last):
File '<interactive input>' line 1, in ?
KeyError: bozo
```

59

Removing Dictionary Entries

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> del d['user']        # Remove one.
>>> d
{'p':1234, 'i':34}
>>> d.clear()            # Remove all.
>>> d
{}
>>> a = [1,2]
>>> del a[1]                  # (del also works on lists)
>>> a
[1]
```

60

Exercise 3: Kid's Height for Slide Use

- Description
  - Develop a program to determine if a child is tall enough to use a specific playground slide based on their height.

61

Exercise 4: Password Generator

- Description
  - Design a tool that generates strong and random passwords for enhanced security.

62

Exercise 5: Password Checker

- Description
  - Create a utility to check the strength and validity of user-generated passwords.

Useful Accessor Methods

```
>>> d = {'user':'bozo', 'p':1234, 'i':34}
>>> d.keys()       # List of current keys
['user', 'p', 'i']
>>> d.values()      # List of current values
['bozo', 1234, 34]
>>> d.items()       # List of item tuples.
[('user','bozo'), ('p',1234), ('i',34)]
```

65

Application of Python in Banking and FIs

- Handling data using Python libraries (NumPy and Pandas).
- Data manipulation and cleaning.
- Data visualization using Matplotlib and Seaborn.
- Exploratory data analysis –using Bank loan default.

66