www.iitkirba.xyz

Python Tuples: An Overview

- A **tuple** is an ordered, immutable collection of items in Python.
- Tuples are similar to lists, but unlike lists, tuples **cannot be changed** once they are created.
- Tuples can contain elements of different data types, such as integers, strings, and even other tuples.

Made with Gamma

www.iitkirba.xyz

Understanding Tuple Basics

Defining Tuples

Tuples are created using parentheses and commas. They are similar to lists, but the elements are immutable - they can't be changed.

Illustrative Examples

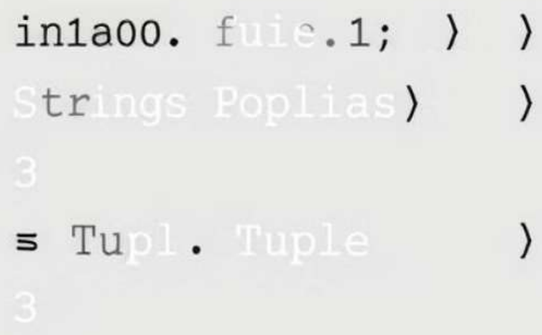
```
my_tuple = (1, 2, 'three')

single_element_tuple = (5,)

empty_tuple = ()
```

Made with Gamma

www.iitkirba.xyz



Key Tuple Characteristics

Ordered

Elements within a tuple maintain their order, accessible through indexing.

Immutable

Once a tuple is created, its elements cannot be modified.

Allow Duplicates

Tuples can contain multiple instances of the same element.

Mixed Data Types

Tuples can hold different data types, such as integers, strings, and even other tuples.

Made with Gamma

www.iitkirba.xyz

Tuple

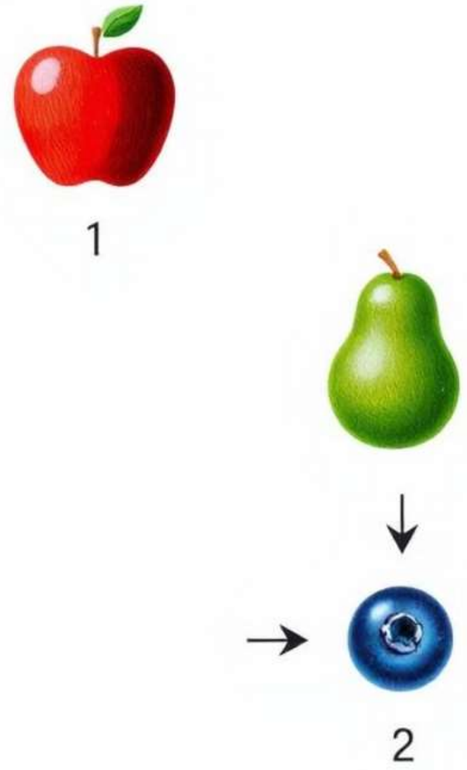
Accessing and Modifying Tuple Elements

Indexing

Use indexing to access individual elements in a tuple, starting from 0.

Immutability

Tuples are immutable, meaning you cannot change their elements directly.



Made with Gamma

www.iitkirba.xyz

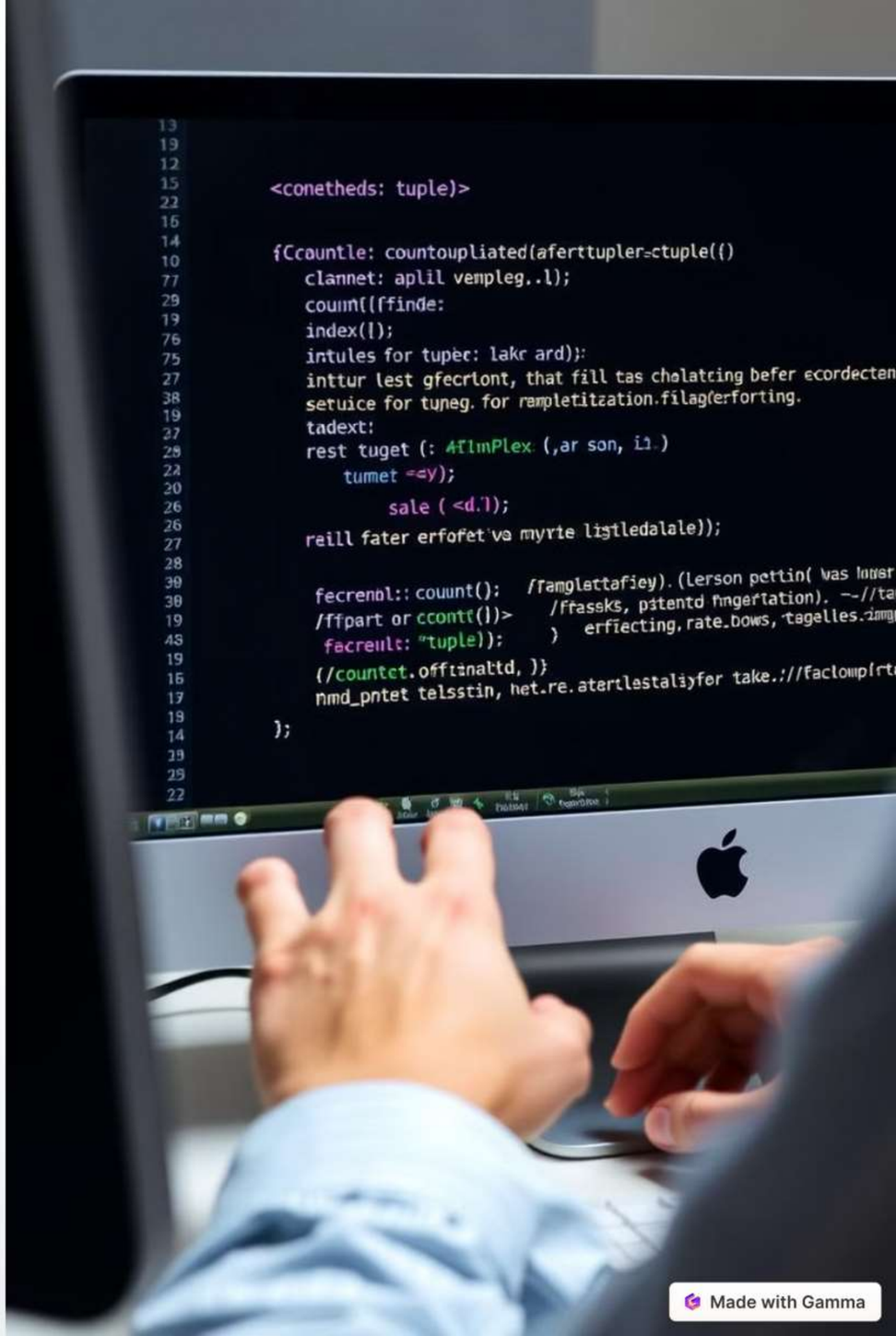
Essential Tuple Methods

count()

Returns the number of occurrences of a specific value within a tuple.

index()

Returns the index of the first occurrence of a value in a tuple.



Made with Gamma

www.iitkirba.xyz

Tuple Operations: Combining and Repeating

Concatenation

Use the + operator to join two tuples, creating a new tuple with all the elements.

Repetition

Use the * operator to repeat a tuple a specified number of times.

Made with Gamma

www.iitkirba.xyz

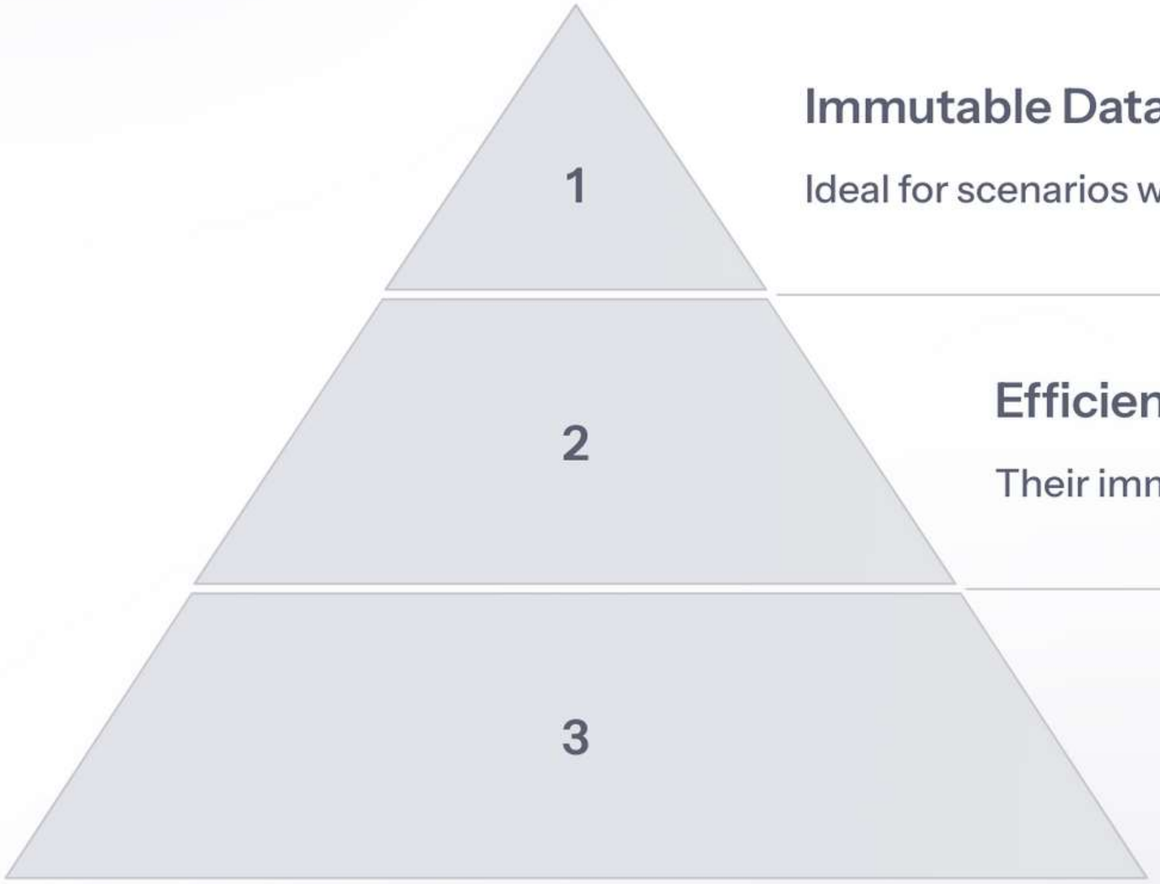
Tuples vs. Lists: Choosing the Right Tool

Feature	Tuple	List
Mutability	Immutable	Mutable
Syntax	(1, 2, 3)	[1, 2, 3]
Methods	Fewer (count, index)	More (append, remove, etc.)
Performance	Faster	Slower

Made with Gamma

www.iitkirba.xyz

Conclusion: Tuples in Action



Immutable Data

Ideal for scenarios where you need to ensure data integrity.

Efficient Performance

Their immutability contributes to faster execution.

Versatile Applications

Used for storing fixed data, returning multiple values, and as dictionary keys.

Made with Gamma

www.iitkirba.xyz

Python Sets: A Concise Overview

.What is a Set?

- A **set** is an unordered collection of **unique** elements in Python.
- Sets are similar to lists and tuples but do not allow duplicate values and do not maintain order.
- Sets are **mutable** (you can add and remove elements), but they are **immutable** in the sense that their elements cannot be modified once added (no indexing or slicing).

Made with Gamma

www.iitkirba.xyz

What is a Set?

Unique Elements

A set is an unordered collection of unique elements. It's like a list or tuple, but it doesn't allow duplicates.

No Order

Sets don't maintain order, so you can't access elements by index like you would with a list.

Made with Gamma

www.iitkirba.xyz

```
caur bracce: Stt.:71){
jnn lat +5t.11al:(50);}
createsed the ceit.[set).it [e-0).].::71;{
    lseelictt.10;;    ;;
    latelscnt.rel:[ittC402) };
}
creasess = in set()
irc /abeut it Sel.80));
caites        .1;;    ;;;
ire isted the selt.'itt.1]; [sett'[ (t).rlet, ;}
    lseclictt.17);   ;;;
    latelscnt.rel:[ittC446) };
}
```

Creating Sets

Curly Braces

You can create a set using curly braces {}.

set() Constructor

Use the `set()` constructor to create a set from an existing iterable, such as a list or tuple.

Made with Gamma

www.iitkirba.xyz

Key Characteristics of Sets

Unordered

Elements in a set have no specific order.

Unique

No duplicate elements are allowed in a set.

Mutable

You can add or remove elements from a set, but you cannot change its elements once added.

No Indexing

Unlike lists, you can't access elements in a set by index.

Made with Gamma

www.iitkirba.xyz

Basic Set Operations

1

add()

Adds a single element to the set.

2

remove()

Removes an element from the set. Raises a `KeyError` if the element doesn't exist.

3

discard()

Removes an element from the set. Does nothing if the element doesn't exist.

4

pop()

Removes and returns a random element from the set.

5

clear()

Removes all elements from the set.

6

len()

Returns the number of elements in the set.

www.iitkirba.xyz

Set Methods

1. **union()**
   Returns a new set containing all elements from both sets.

2. **intersection()**
   Returns a new set containing only the elements that are in both sets.

3. **difference()**
   Returns a new set containing elements that are in the first set but not in the second.

4. **symmetric_difference()**
   Returns a new set with elements in either of the sets, but not in both.

5. **issubset()**
   Checks if all elements of one set are in another set.

6. **issuperset()**
   Checks if the set contains all elements of another set.

7. **isdisjoint()**
   Checks if two sets have no elements in common.

8. **copy()**
   Creates a shallow copy of the set.

www.iitkiba.xyz

Basic Set Operations

1

add()

Adds a single element to the set.

2

remove()

Removes an element from the set. Raises a 'KeyError' if the element doesn't exist.

3

discard()

Removes an element from the set. Does nothing if the element doesn't exist.

4

pop()

Removes and returns a random element from the set.

5

clear()

Removes all elements from the set.

6

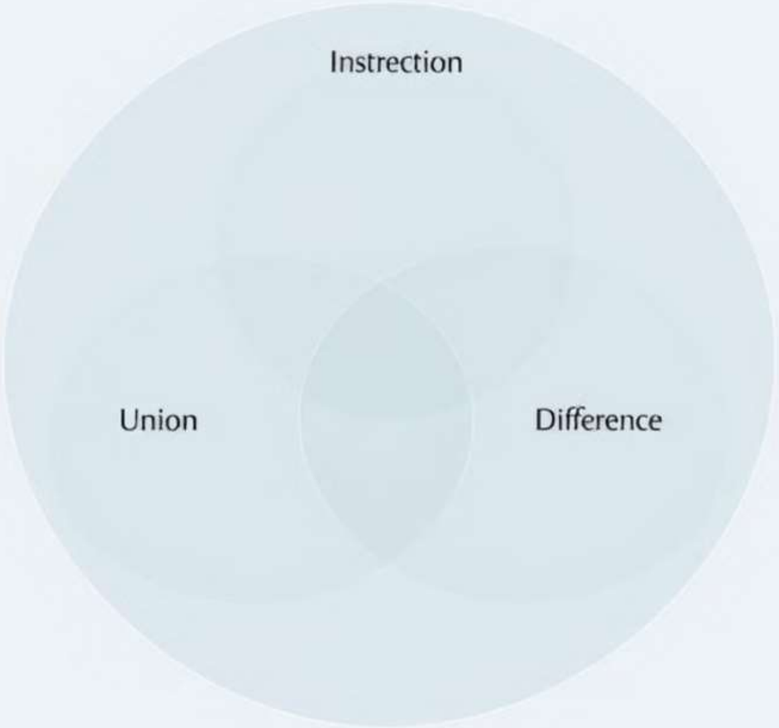
len()

Returns the number of elements in the set.

www.iitkirba.xyz

Set Operations with Operators

Operator	Operation	Example
|	Union	set1 | set2
&	Intersection	set1 & set2
-	Difference	set1 - set2
^	Symmetric Difference	set1 ^ set2



Made with Gamma

www.iitkirba.xyz

Conclusion

Sets offer a powerful way to work with unique elements in Python. They provide efficient operations for managing data, eliminating duplicates, and performing mathematical set operations. Remember, sets are unordered and mutable, making them ideal for situations where order is not essential.

Made with Gamma

www.iitkirba.xyz

Understanding Python Dictionaries

This presentation will delve into the fascinating world of Python dictionaries, a fundamental data structure that provides a powerful tool for organizing and accessing information within your programs. We'll explore the key features, capabilities, and best practices of dictionaries, leaving you equipped to confidently incorporate them into your Python projects.

Made with Gamma

www.iitkirba.xyz

The Basics of Python Dictionaries

Dictionaries store data in key-value pairs, offering a structured way to map information. Each key serves as a unique identifier for its associated value, which can be of any data type.

Key Properties

Keys must be immutable, ensuring their integrity and uniqueness. This means they can be strings, numbers, or tuples.

Value Properties

Values can be of any data type, including strings, numbers, lists, or even other dictionaries.

Made with Gamma

www.iitkirba.xyz

```
corete tat Lite that bat; acth;
port out traftentasr bat picitalBittile ),
prtect but be istule ietect (ng)
)  Car bat
 brattof Litiler sist(or) , ....:
 bhoth mrCiteFattocrLPattlen )
```

Creating and Accessing Dictionaries

Let's explore the fundamental operations of creating and accessing items within a dictionary.

Creation

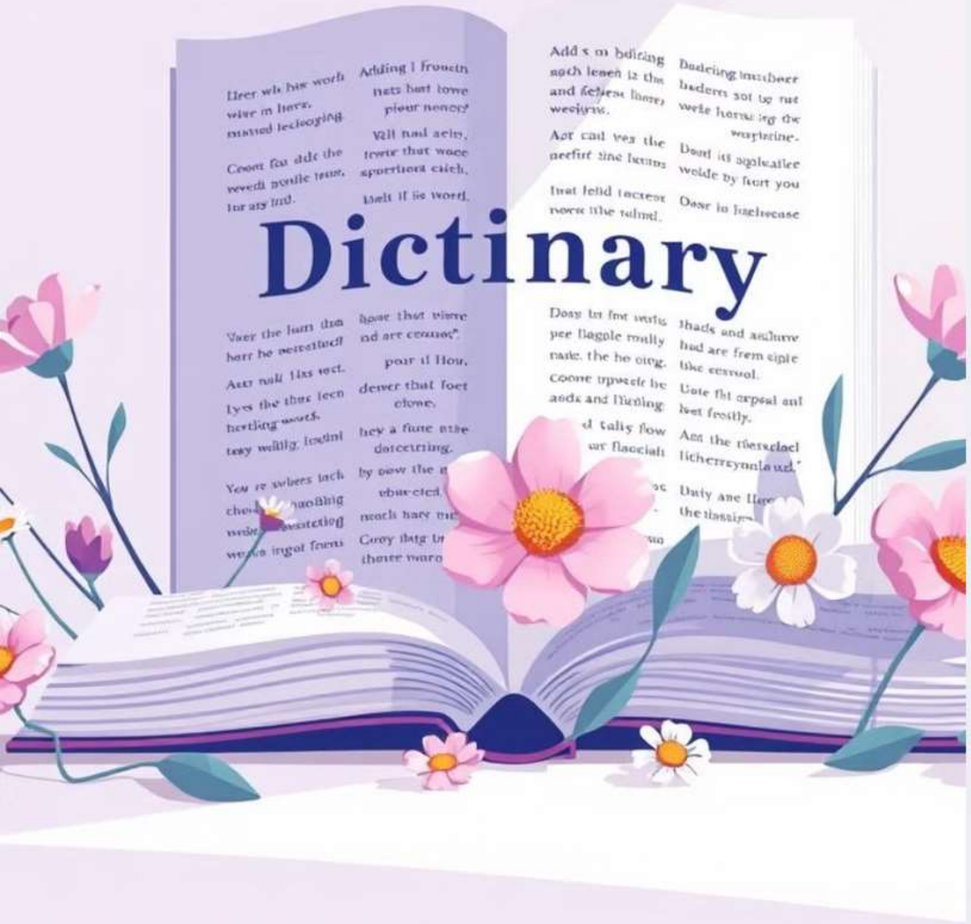
Dictionaries are created using curly braces { } and key-value pairs separated by colons. Example: my_dict = { "brand": "Ford", "model": "Mustang", "year": 1964 }

Accessing Values

To access a value, use the key name within square brackets. Example: car_model = my_dict["model"]

Made with Gamma

www.iitkirba.xyz



Exploring Dictionary Features

Dictionaries are highly dynamic and offer a range of methods for manipulating their content.

1 Changeable

You can modify, add, or remove items in a dictionary after its creation.

2 Ordered

From Python 3.7 onwards, dictionaries maintain the order in which items were inserted.

3 No Duplicates

A dictionary cannot have two items with the same key. If a duplicate key is added, it overwrites the existing one.

Made with Gamma

www.iitkirba.xyz

Key Methods for Dictionary Manipulation

Here are some essential methods that enable you to work effectively with dictionaries.

keys()

Returns a list of all keys in the dictionary.

values()

Returns a list of all values in the dictionary.

items()

Returns a list of key-value pairs as tuples.

update()

Updates the dictionary with items from a given argument.

Made with Gamma

www.iitkirba.xyz

Looping Through Dictionaries

Looping through dictionaries allows you to access and process each key-value pair.

1

Looping Through Keys

The for loop iterates over the keys of the dictionary. Example: for x in thisdict: print(x)

2

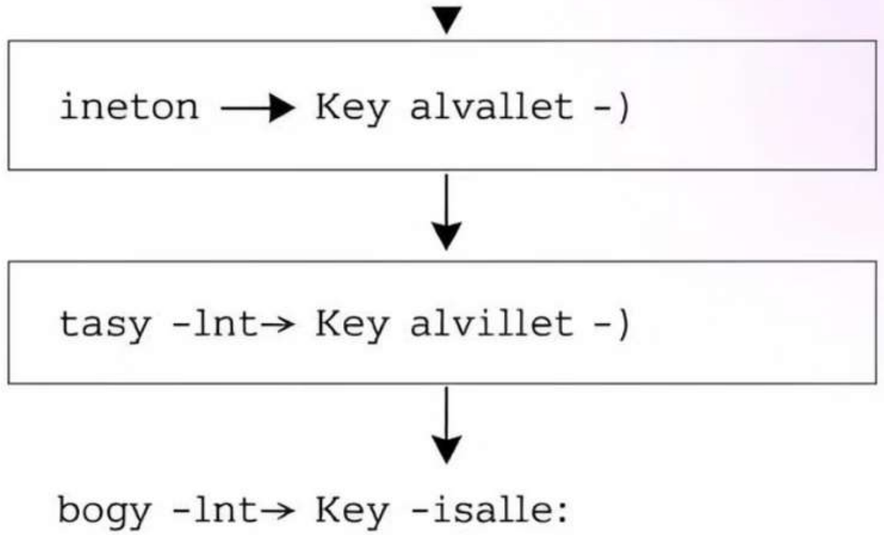
Looping Through Values

Access the values using the key within the loop. Example: for x in thisdict: print(thisdict[x])

3

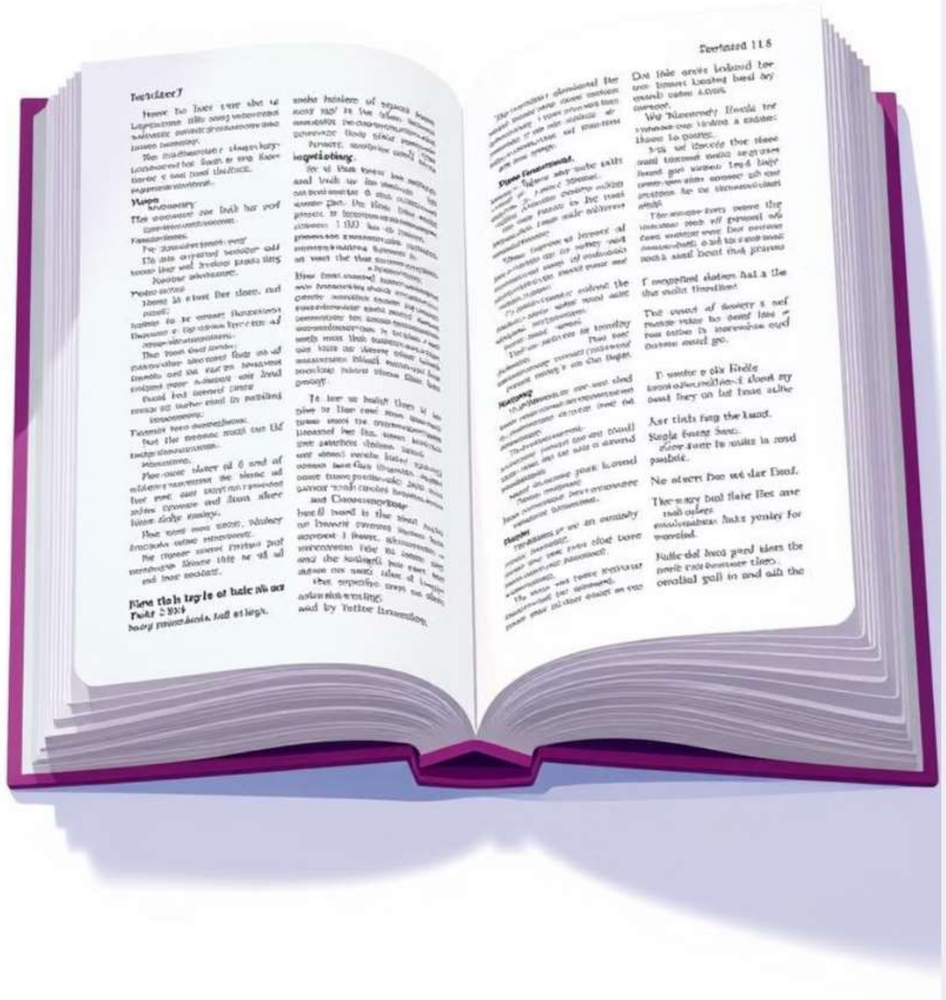
Looping Through Items

The items() method returns key-value pairs as tuples. Example: for x, y in thisdict.items(): print(x, y)



Made with Gamma

www.iitkirba.xyz



Copying Dictionaries

To avoid unintended modifications to the original dictionary, it's important to create a copy before making changes.

1

Shallow Copy

The copy() method creates a shallow copy, where changes in the original dictionary may affect the copy if they involve mutable values.

2

Deep Copy

To create a truly independent copy, use the deepcopy() function from the copy module.

Made with Gamma

www.iitkirba.xyz

Nested Dictionaries: Organizing Data Hierarchically

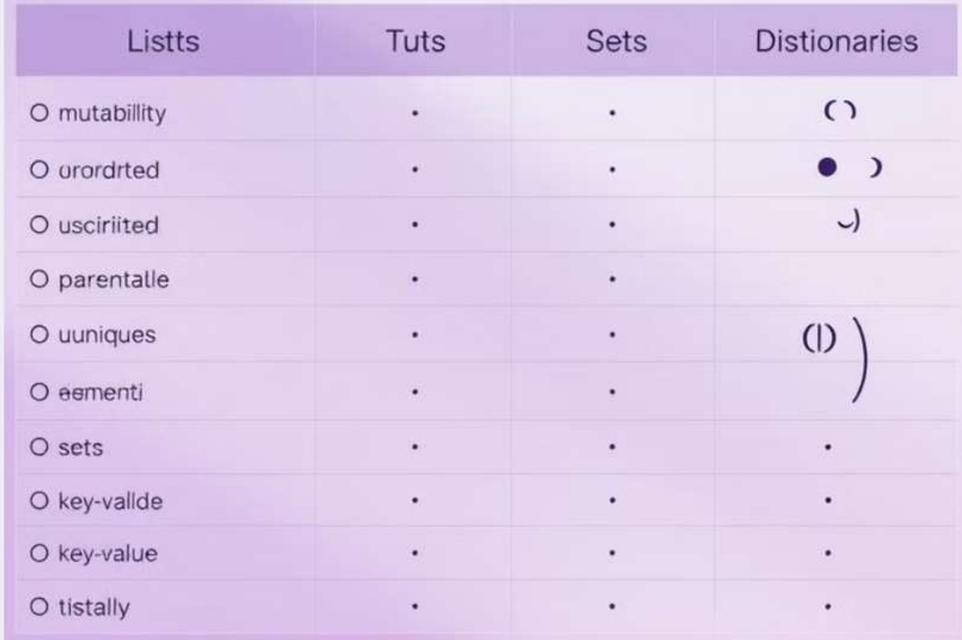
Nested dictionaries allow you to represent complex data structures, such as family trees or organizational hierarchies.

www.iitkirba.xyz

Understanding Key Differences

Let's compare dictionaries with other fundamental data structures.

Data Structure	Mutability	Order	Duplicates
List	Mutable	Ordered	Allowed
Tuple	Immutable	Ordered	Allowed
Set	Mutable	Unordered	Not Allowed
Dictionary	Mutable	Ordered (Python 3.7 and above)	Not Allowed (for keys)



Made with Gamma

www.iitkirba.xyz

Conclusion: Mastering Python Dictionaries

By understanding Python dictionaries, you've gained a powerful tool for organizing and manipulating data. Use these key takeaways to confidently apply dictionaries in your Python projects.

1

Flexibility

Dictionaries offer a flexible and dynamic way to store and access data.

2

Efficiency

Key-value pairs enable quick access to specific data, making dictionaries efficient for data retrieval.

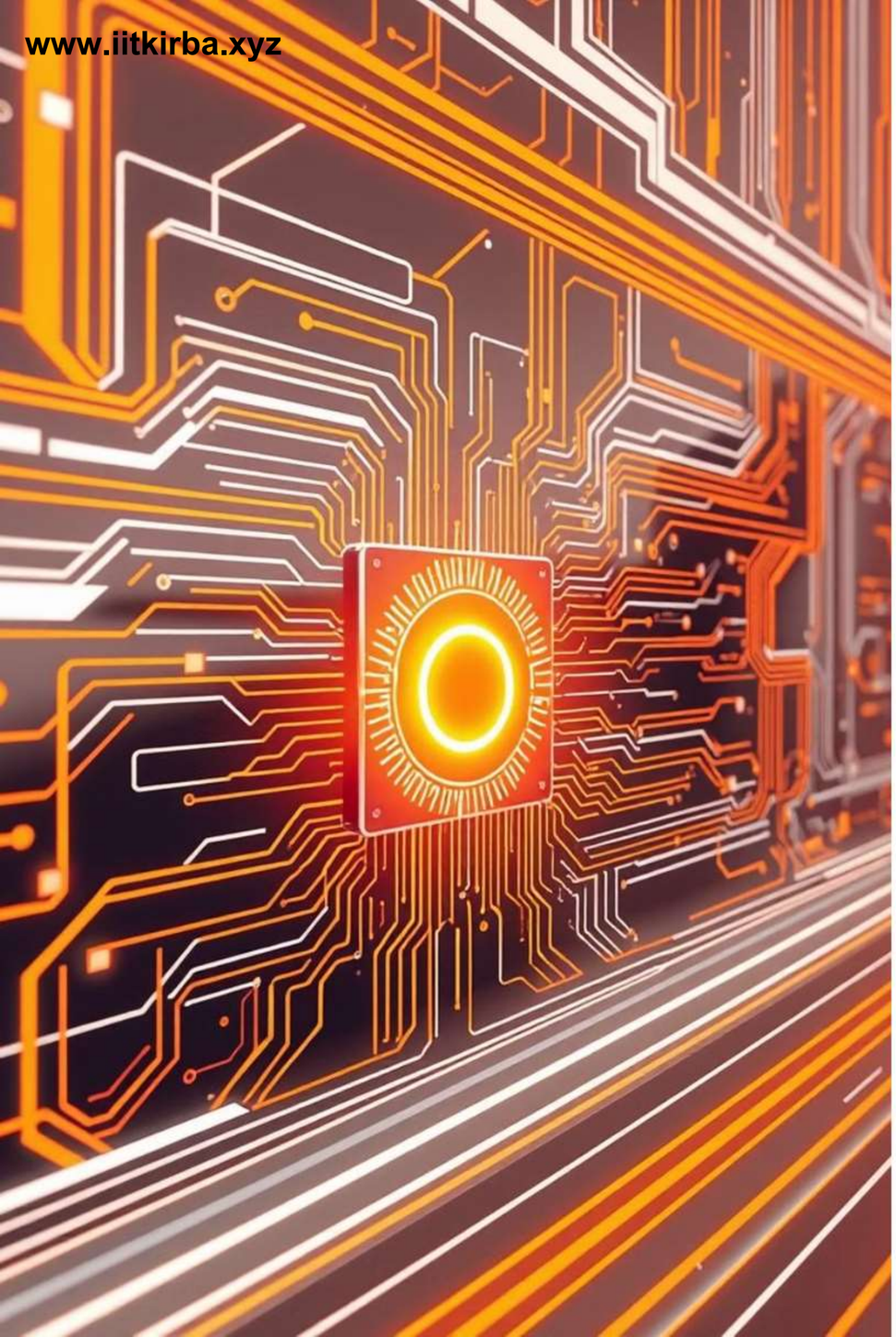
3

Organization

Dictionaries excel at structuring and organizing complex data relationships.

Made with Gamma

www.iitkirba.xyz



Unlocking the Power of Python Functions

Made with Gamma

www.iitkirba.xyz

The Essence of Python Functions

Code Blocks

Python functions are blocks of code that perform a specific task, enabling code reuse and organization.

Reusability and Efficiency

Instead of writing the same code repeatedly, functions allow us to call them multiple times, saving time and effort.

Made with Gamma

www.iitkirba.xyz

Benefits of Function Utilization

1 Increased Code Readability

Functions break down complex code into manageable units, making it easier to understand and maintain.

2 Enhanced Code Reusability

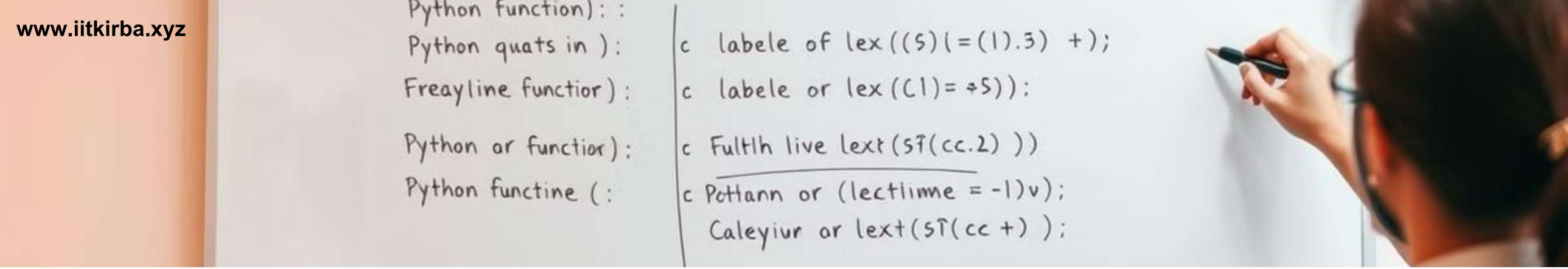
Functions allow us to reuse code blocks across different parts of our program, promoting modularity.

3 Improved Code Efficiency

Functions streamline code by reducing redundancy, leading to shorter and more efficient programs.

Made with Gamma

www.iitkirba.xyz



Defining Python Functions

The syntax for declaring a function in Python uses the "def" keyword, followed by the function name, parentheses, and a colon. The code block within the function is indented.

```
def function_name(parameter: data_type) -> return_type:
  """Docstring"""

  # body of the function

  return expression
```

www.iitkirba.xyz

Types of Functions in Python

Built-in Library Functions

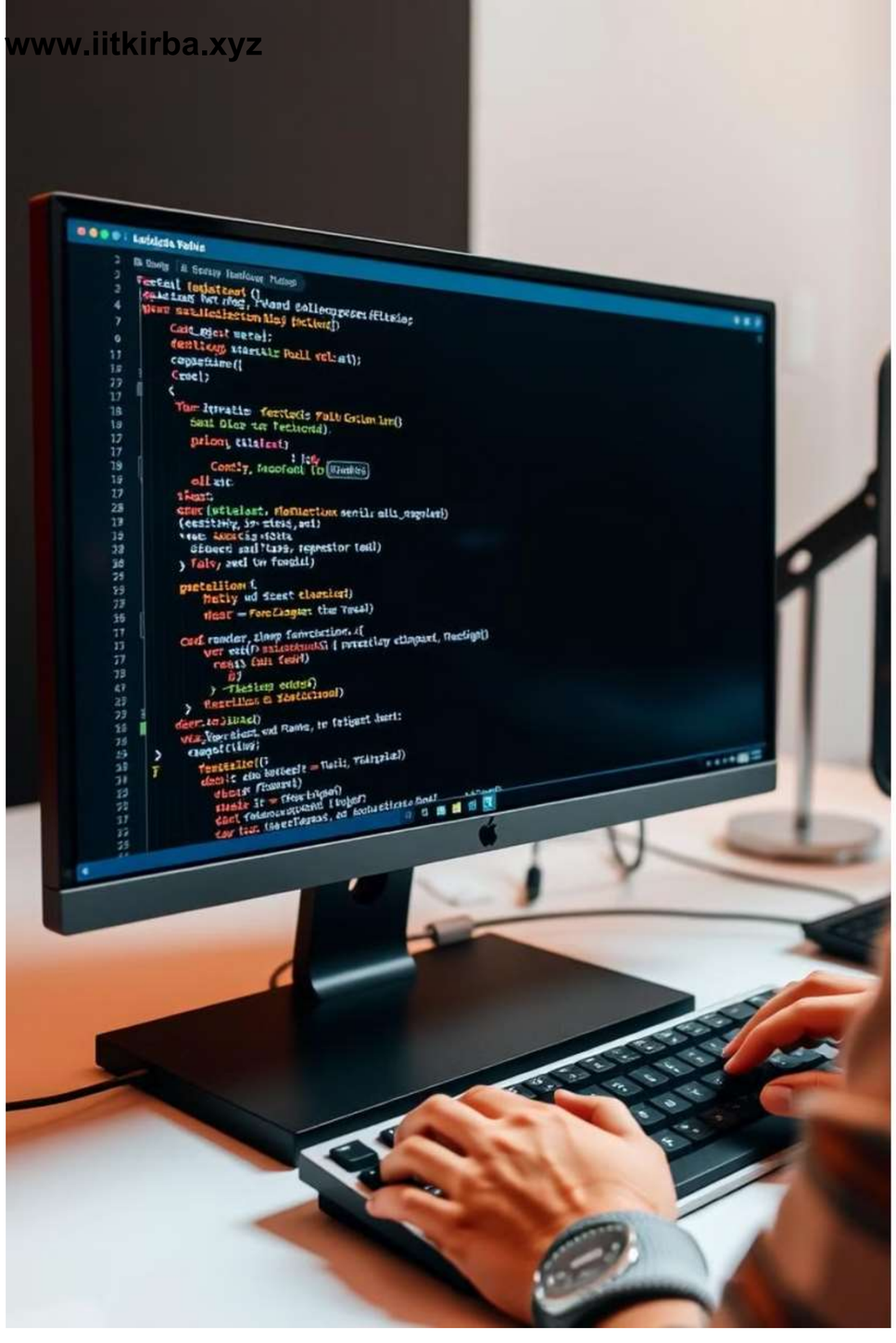
Standard functions readily available in Python's libraries for common tasks (e.g., len(), print(), input()).

User-Defined Functions

Functions created by programmers to perform specific tasks according to their requirements.

Made with Gamma

www.iitkirba.xyz



Creating a Python Function

We can define functions using the "def" keyword and add desired functionalities. Let's look at a simple example to illustrate the process.

```
def fun():
  print("Welcome to GFG")
```

Made with Gamma

www.iitkirba.xyz

Calling a Python Function

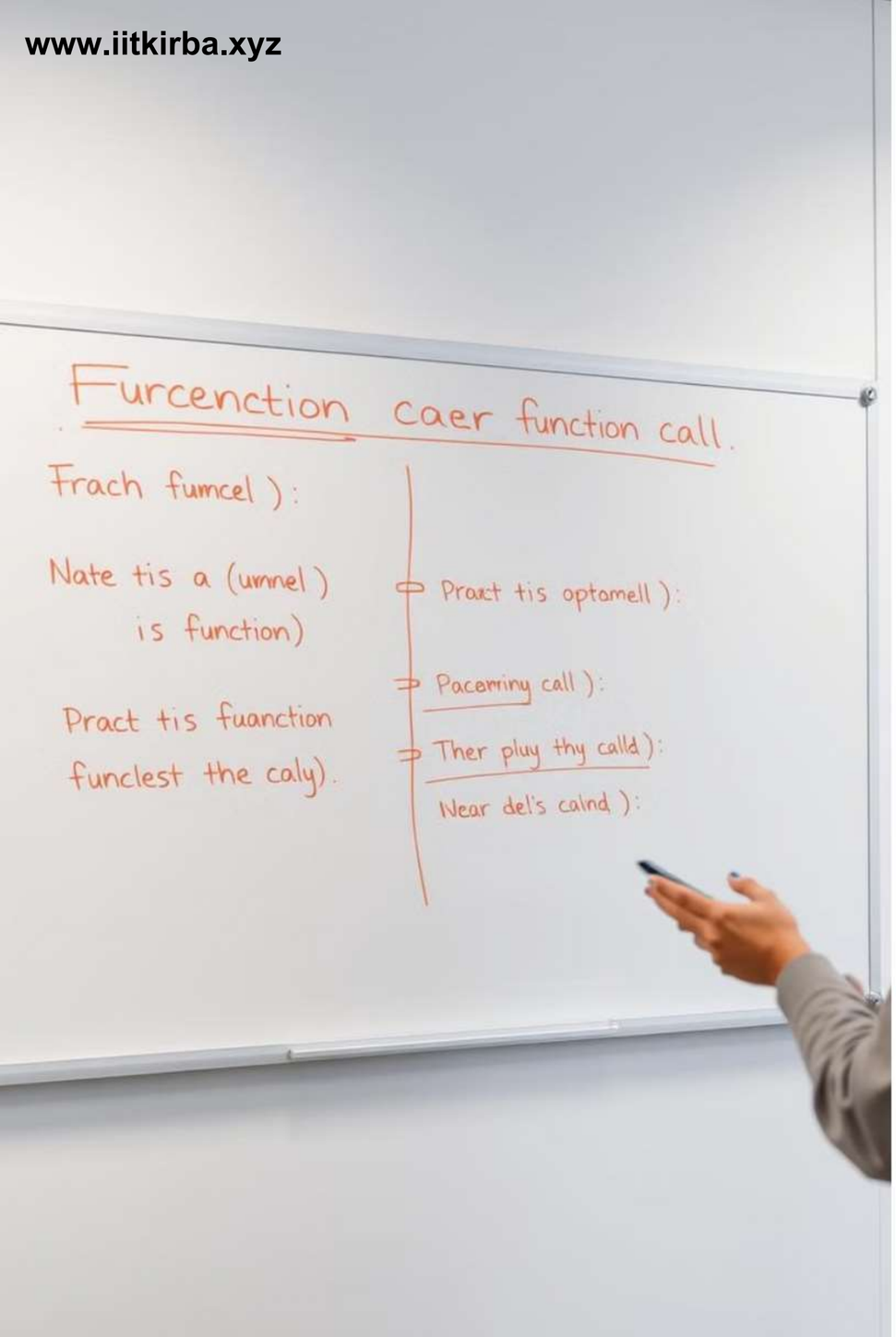
To execute a defined function, we call it by its name followed by parentheses, potentially including parameters.

```
def fun():
  print("Welcome to GFG")

# Driver code to call the function
fun()
```

Made with Gamma

www.iitkirba.xyz



Functions with Parameters

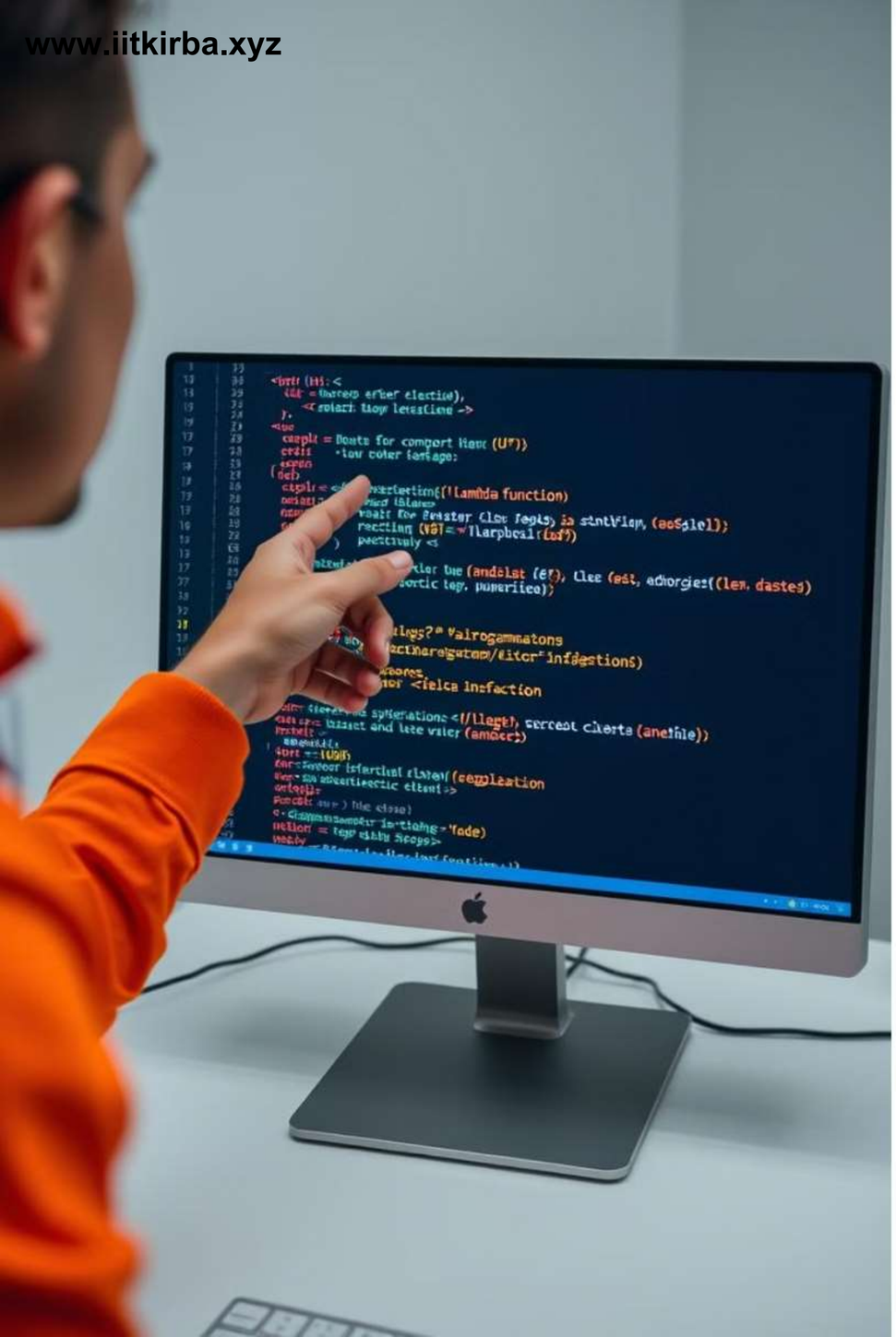
Functions can accept parameters, which are values passed during function calls. These parameters act as input variables within the function.

```
def add(num1: int, num2: int) -> int:
  """Add two numbers"""
  num3 = num1 + num2
  return num3

# Driver code
num1, num2 = 5, 15
ans = add(num1, num2)
print(f"The addition of {num1} and {num2} results {ans}.")
```

Made with Gamma

www.iitkirba.xyz



Anonymous Functions: Lambda Expressions

Lambda expressions provide a concise way to define anonymous functions, which are functions without a name. They are often used for short, simple operations.

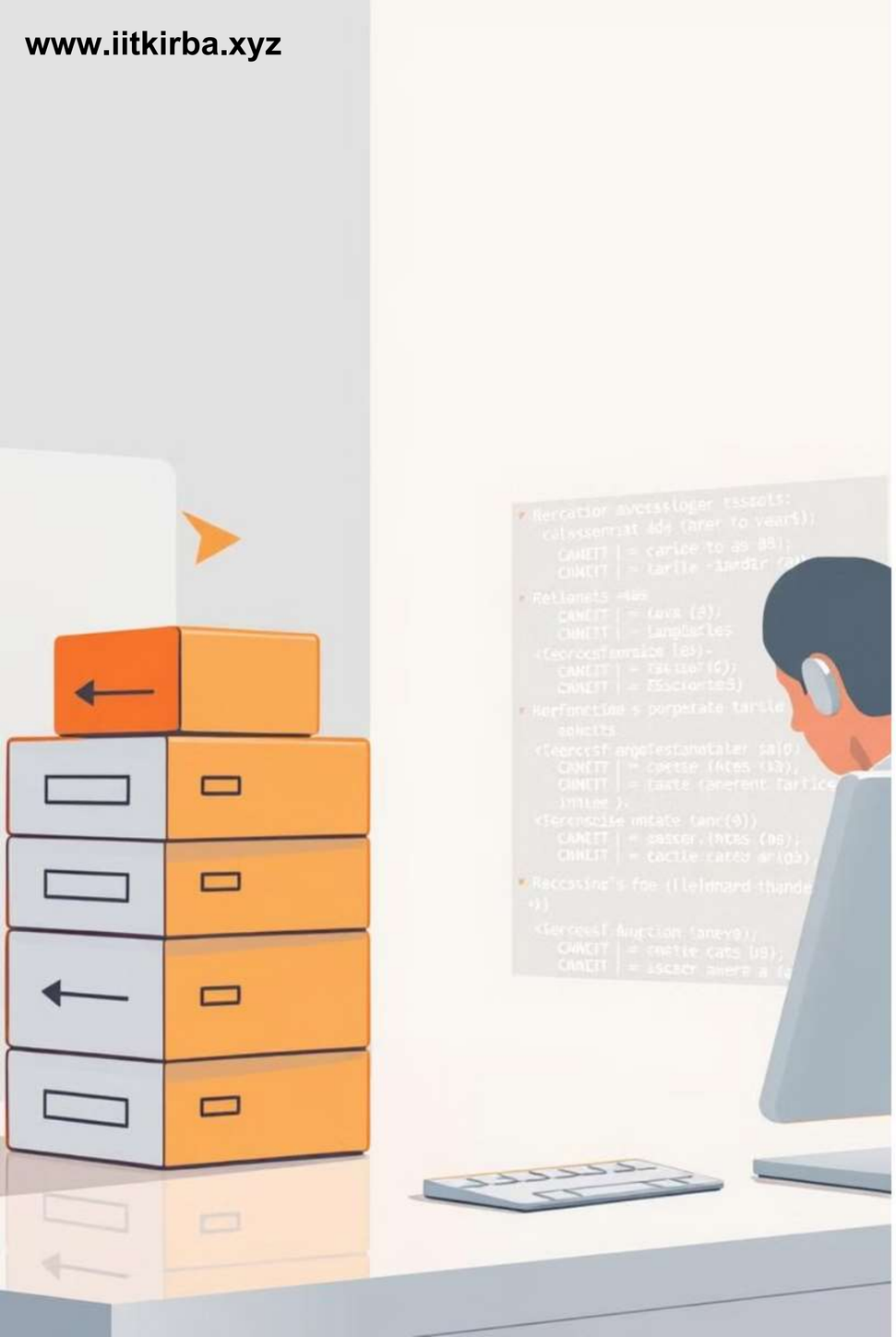
```
# Python code to illustrate the cube of a number using a lambda function
def cube(x):
  return x * x * x

cube_v2 = lambda x: x * x * x

print(cube(7))
print(cube_v2(7))
```

Made with Gamma

www.iitkirba.xyz



Recursive Functions in Python

Recursive functions call themselves, providing a solution for problems that can be broken down into smaller, similar subproblems. Careful use is crucial to avoid infinite loops.

```
def factorial(n):
  if n == 0:
    return 1
  else:
    return n * factorial(n - 1)

print(factorial(4))
```

Made with Gamma

Topic

1.Passing Arguments in Python
- •Positional, Keyword, Default, Variable-Length

2.Anonymous Functions (*lambda*)
- •Syntax, Use Cases, Examples

3.Recursive Functions
- •Concept, Examples, Benefits & Drawbacks

www.iitkirba.xyz

Passing Arguments

What are Function Arguments in Python?

- **Definition:** Arguments are the values passed to a function when it is called.
- **Purpose:** They allow functions to accept input and customize behavior.

www.iitkirba.xyz

Passing Arguments

- **Types of Arguments**
- **Positional Arguments:**
  - Passed in the correct order.
  - Example:

```
def greet(name, age):
print(f"Hello, {name}. You are {age} years old.")
greet("Alice", 25)
```

- **Keyword Arguments:**
- Passed with parameter names.
- Example:

```
greet(age=25, name="Alice")
```

www.iitkirba.xyz

Passing Arguments

- **Default Arguments**
- **Definition:** Assign default values to parameters.
- **Example:**

```
def greet(name, age=30):
print(f"Hello, {name}. You are {age} years old.")
greet("Alice")  # Uses default age
```

Passing Arguments

➢Variable-Length Arguments

- ***args (Non-keyword Arguments):**

•Collects additional positional arguments.
•Example:

```
def add_numbers(*args):
  return sum(args)
print(add_numbers(1, 2, 3))
# Output: 6
```

- ****kwargs (Keyword Arguments):**

Collects additional keyword arguments.
Example:

```
def show_info(**kwargs):
for key, value in kwargs.items():
  print(f"{key}: {value}")
show_info(name="Alice",  age=25)
```

www.iitkirba.xyz

Anonymous Functions (Lambda Functions)

- What is a Lambda Function?

•**Definition:** A lambda function is a small, anonymous function defined using the lambda keyword.
•**Syntax:**

```
lambda arguments: expression
```

•**Features:**Can have multiple arguments but only one expression.
•Useful for short, simple operations.

www.iitkirba.xyz

Anonymous Functions (Lambda Functions)

- Examples of Lambda Functions

  Basic Example

  ```
  square = lambda x: x * x
  print(square(5))  # Output: 25
  ```

  With Multiple Arguments:

  ```
  add = lambda x, y: x + y
  print(add(3, 7))  # Output: 10
  ```

www.iitkirba.xyz

Anonymous Functions (Lambda Functions)

- Use Cases of Lambda Functions

  Sorting

```
names = ["Alice", "Bob", "Charlie"]
names.sort(key=lambda x: len(x))
print(names)  # Output: ['Bob', 'Alice', 'Charlie’]
```

  Mapping:

```
numbers = [1, 2, 3, 4]
squares = map(lambda x: x ** 2, numbers)
print(list(squares))  # Output: [1, 4, 9, 16]
```

www.iitkirba.xyz

Recursive Functions

- **What is Recursion?**
- **Definition:** A recursive function is a function that calls itself to solve a smaller instance of the same problem.
- **Structure:**
  - Base Case: Terminates the recursion.
  - Recursive Case: Calls itself.

www.iitkirba.xyz

Recursive Functions

- Example of a Recursive Function
- Factorial Calculation:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
print(factorial(5))  # Output: 120
```

Explanation:

- Base case: n == 0.
- Recursive step: n * factorial(n-1).

www.iitkirba.xyz

Recursive Functions

- **Benefits and Drawbacks of Recursion**
- **Benefits:**
  - Elegant and concise code.
  - Useful for problems like tree traversal, backtracking, etc.
- **Drawbacks:**
  - Higher memory usage (stack).
  - Can lead to stack overflow if the base case is not reached.