Foundation Level

Chapter 1

1.1. Python Basics

Why Python?

- Python is one of the most popular and beginner-friendly programming languages.
- Python is versatile: used in web development, data science, AI, automation, and more.
- Huge community and lots of libraries to help you build almost anything.

1.2. Variables

A variable is like a box with a name.

- You put something inside the box (a number or word).
- The name on the box helps you find what's inside later.

Variable Naming Rules:

1. **Start with a letter or underscore (_)**

   Example: name, _score

2. **Not allowed: 1age (can't start with a number)** Can

   contain letters, numbers, and underscores Example:

   age1, student_name

3. **No spaces allowed**

   Use underscore _ instead of space

Example: first_name (correct), first name (wrong)

4. Cannot use Python reserved keywords

Words like if, for, while, class cannot be variable names

5. Variable names are case-sensitive

Age and age are different variables

6. Keep names meaningful

Use names that describe what the variable holds, like score or student_age

Example:

name = "Sprout" student

= 900

- name box holds "Sprout".
- student box holds 900.

You can use the box's name to get the value: print ("Hello, " + name)

Output:

Hello, Sprout

How to Initialize a Variable Using Assignment Operator

- Choose a variable name
- Use the assignment operator = to give it a value.

Syntax: Variable_name

= value **Example:**

name = "Sprout"

location = "Coimbatore"

age = 6

Explanation:

• Stores the word "Sprout" in the variable name • Stores the word "Coimbatore" in the variable location • Stores number 6 in the variable age

AI Tutor Questions

4C Skill	AI Tutor Question
Critical Thinking	What might happen if you reuse a variable name by accident? Try and observe.
Creativity	Create a profile card with name, age, and favorite color using variables.
Communication	Explain to a friend why variable names shouldn't start with a number.
Collaboration	Work with a peer: one writes code for name and age, another adds hobbies.

1.3. Data Types

What Are Data Types?

• When we create a variable, Python needs to know what kind of value is stored inside.

• That “kind” is called a data type.

Imagine a school bag:

• If you put books in it → it becomes a book bag.

• If you put snacks → it becomes a lunch bag.

In Python, the type of value decides the data type of the variable.

Data Type	Description	Example
int	Whole numbers	5, -10, 0
float	Decimal numbers	3.14, -0.5
bool	True/False values	True, False
str	Text (string) data	"Hello", '123'

Syntax:

variable_name = value

1.3.1. Integer (int)

What is an Integer?

An integer is a whole number, no decimals. It

can be:

- Positive → 5, 100
- Negative → -7, -22
- Zero → 0

Syntax

variable_name = integer_value

Example Program

```
age = 5

year = 2020

print("Age:", age)

print("Year:", year)
```

Explanation

- age = 5 → We're storing a whole number in the variable age. That's an **integer**.
- year = 2020 → Years are whole numbers, so again this is int.

We use print() to display them.

AI Tutor Questions:

4C Skill	AI Tutor Question
Critical Thinking	Why would using a float instead of an integer cause issues when you're counting people in a classroom?
Creativity	Design a small student info card using integers like roll_number, age, and grade. Print the values and share it.
Communication	If a user enters their age as a float (e.g., 10.5), how would you explain to them why the system needs an integer?
Collaboration	Work with a friend to create a program that accepts a student's roll number and age, then prints a welcome message.

1.3.2. Float (Decimal Numbers)

What is a Float?

A float is a number that has a decimal point.

Examples: 3.14, 0.5, -7.25

We use floats when we need precision , like in money, temperature, or measurements.

Syntax

variable_name = decimal_value

Example Program

price = 49.99

temperature = -2.5

height = 5.8

print("Price:", price)

print("Temperature:", temperature)

print("Height:", height) **Explanation**

- price = 49.99 → This is a **float** because it has a decimal. Useful when dealing with money.
- temperature = -2.5 → Even though it's negative, it's still a float because of the decimal point.
- height = 5.8 → Height is often measured in points or inches, so float makes sense.

AI Tutor Questions:

4C Skill	AI Tutor Question
CriticalThinking	What problems might occur if you use an integer instead of a float for calculating someone's body temperature?

Creativity	Create a mini shopping list using float values like price = 49.99, discount = 5.5. Calculate and print the final price.
Communication	Explain to someone new to programming why 3.0 and 3 are stored differently in memory.
Collaboration	Pair up and create a weather report generator that stores today's temperature, humidity, and wind speed using floats.

1.3.3. Boolean (True or False)

What is a Boolean?

A Boolean is a special data type that has only two possible values:

- True
- False

We use it for decisions, like:

- Is a student present?
- Has the user logged in?
- Is the number greater than 10?

Syntax variable_name

= True or

variable_name = False

Example Program

is_logged_in = True

is_holiday = False

passed_exam = True

print("User logged in:", is_logged_in) print("Is

today a holiday?", is_holiday) print("Passed the

exam:", passed_exam) **Explanation**

• is_logged_in = True → The user has logged in, so it's True. • is_holiday = False → Today is not a holiday, so we mark it as False. • passed_exam = True → This student passed, so we store it as True.

AI Tutor Questions:

4C Skill	AI Tutor Question
Critical Thinking	How does using Boolean values help in deciding whether a user should be allowed to access a page or not?
Creativity	Write a program that checks if someone is eligible for a prize based on has_coupon = True. What other conditions?
Communication	How would you explain the difference between using = and == when working with Boolean conditions?
Collaboration	Work with a peer to write a login check. Store is_logged_in = False and later change it to True. Show status.

1.4. Strings

What is a String?

- A string is simply a sequence of characters — like words or sentences inside quotes. It can contain letters, numbers, and symbols.
- You can think of a string like a sentence written on a ribbon — each character has a place (called an index), and you can pull characters out or change them around.

Creating a String

Syntax:

text = "sprout"

or

text = ' Brighter tomorrow'

Example:

greeting = "Welcome to sprout" print(greeting)

Output:

Welcome to sprout

Explanation:

- The variable greeting stores a message (string).
- print() displays it.

String Indexing

Each character in a string has a position number (index), starting from 0. **Example:**

word = "Sprout" print(word[0])

//First letter print(word[5])

//Last letter

Output:

S

t

Explanation:

- word[0] gives the first character.
- word[5] gives the sixth character (index starts from 0).

String Slicing

You can cut out parts of a string using slicing.

Syntax:

string[start:end]

Example:

msg = "HelloSprout" print(msg[0:5])

From index 0 to 4 **Output:**

Hello

Explanation:

- Starts from index 0 and goes up to (but not including) index 5.

Common String Methods

Python gives many built-in methods to work with strings.

Method	**Description**
lower()	Converts to lowercase

upper()	Converts to uppercase
strip()	Removes spaces at ends
replace()	Replaces part of the string
split()	Breaks string into a list
find()	Finds the index of a character
len()	Returns length of string

Example Program Using String Methods

name = " Python Programming "

1. print(name.strip()) //Removes spaces

2. print(name.lower()) //Lowercase

3. print(name.upper()) //Uppercase

4. print(name.replace("Python", "Java")) //Replace word 5. print(len(name)) // Count characters including spaces **Output:**

1. Python Programming

2. python programming

3. PYTHON PROGRAMMING

4. Java Programming

5. 24

f-Strings (Formatted Strings)

Used to combine variables with strings in a neat way.

Example:

name = "Sprout"

age = 5

print(f"My name is {name} and I am {age} years old.")

Output:

My name is sprout and I am 5 years old.

Explanation:

• The f before the string lets you place variables inside curly braces {}.

AI Tutor Questions

4C Skill	**AI Tutor Question**
Critical Thinking	What will happen if you try to add a string and an integer directly? Try it and fix the error.
Creativity	Write a creative welcome message by combining 3 different string variables.
Communication	Teach a peer how escape characters like \n and \t work using examples.
Collaboration	Collaborate with a friend to build a small story using multiple strings joined together creatively.

1.5. Operators

Operators are special symbols that perform operations on variables and values. • Think of them as action signs like + for addition or == for checking if two things are equal.

Types of Operators:

1. **Arithmetic Operators**

Used for basic math operations:

- + Addition
- Subtraction
- Multiplication
- / Division
- % Modulus (remainder)
- ** Exponent (power)
- // Floor division (division rounded down)

Syntax

- a + b # adds two numbers
- a - b # subtracts b from a
- a * b # multiplies a and b
- a / b # divides a by b (result can be decimal)
- a % b # gives remainder when a is divided by b
- a ** b # raises a to the power of b
- a // b # divides a by b and gives the whole number part only

Example programs

x = 15

y = 4

Output = 19

Explanation:

- Think of addition as putting two numbers together. Here, 15 plus 4 makes **19**. So, addition holds **19.**

- subtraction means you take one number away from another. 15 minus 4 leaves 11, so subtraction is 11.
- multiplication is like adding a number many times. 15 times 4 is 60, so multiplication stores 60.
- division is splitting a number into equal parts. 15 divided by 4 is 3.75 — not a whole number, so you get a decimal.
- modulus gives the leftover after dividing. When 15 is divided by 4, the leftover (remainder) is 3. So modulus is 3.
- exponent means power — raising a number to another's power. 15 to the power of 4 (15^4) is 50625, so exponent holds 50625.
- floor_division divides and keeps only the whole number part, ignoring decimals. 15 divided by 4 is 3.75, but floor division gives 3.

2. **Comparison Operators**

Used to compare two values and return True or False:

- == Equal to
- != Not equal to
- Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to

Syntax:

- a == b # checks if a is equal to b
- a != b # checks if a is not equal to b
- a > b # checks if a is greater than b
- a < b # checks if a is less than b

- a >= b # checks if a is greater than or equal to b
- a <= b # checks if a is less than or equal to b

Example program:

x = 10

y = 20

- equal = (x == y)
- not_equal = (x != y)
- greater = (x > y)
- less = (x < y)
- greater_equal = (x >= y)
- less_equal = (x <= y)

Explanation:

- equal checks if 10 is the same as 20. Since they are different, it is **False**. • not_equal checks if 10 is different from 20. Since they are different, it is **True**.
- greater checks if 10 is bigger than 20. It's not, so it is **False**. • less checks if 10 is smaller than 20. It is, so it is **True**.
- greater_equal checks if 10 is bigger or the same as 20. It's not, so **False**. • less_equal checks if 10 is smaller or the same as 20. Since 10 is smaller, it is **True**.

3. **Logical Operators**

Used to combine conditional statements:

- and Both conditions True
- or At least one condition True

- not Reverses the result

Syntax:

- a and b → True if both a and b are True
- a or b →True if either a or b is True
- not a →True if a is False, and False if a is True (it reverses)

Example:

x = True

y = False

- and_result = x and y
- or_result = x or y
- not_result = not x

Explanation:

- and_result asks: "Are both x and y True?"

  Here, x is True but y is False, so the answer is **False** because both must be True for and to be True.

- or_result asks: "Is either x or y True?"

  Since x is True, it doesn't matter what y is — or says **True** if any one is True.

- not_result flips the value of x.

Since x is True, not x becomes **False**. It's like saying "not True" = False.

4. **Assignment Operators**

Used to assign or update values:

➢ = Simple assignment

➢ += Add and assign

➢ -= Subtract and assign

➢ *= Multiply and assign

➢ /= Divide and assign

➢ %= Modulus and assign

Syntax:

• a = b →Assign value of b to a

• a += b → Add b to a, then assign result to a (a = a + b)

• a -= b → Subtract b from a, then assign to a (a = a - b) • a *= b →Multiply a by b, then assign to a (a = a * b)

• a /= b →Divide a by b, then assign to a (a = a / b)

• a %= b →Take modulus of a by b, then assign to a (a = a % b) • a **= b → Raise a to power b, then assign to a (a = a ** b) • a //= b → Floor divide a by b, then assign to a (a = a // b) **Example program a = 10**

• a += 5

Now a becomes 10 + 5 = 15

• a -= 3

Now a becomes 15 - 3 = 12

• a *= 2

Now a becomes 12 * 2 = 24

• a /= 4

Now a becomes 24 / 4 = 6.0

• a %= 4

Now a becomes 6.0 % 4 = 2.0

Explanation:

• a = 10 means we start by giving the value 10 to variable a. • a += 5 means "add 5 to a" and update a with this new value, so a becomes 15.

• a -= 3 means "subtract 3 from a" and update a, so now a is 12. • a *= 2 means "multiply a by 2" and update a to 24.

• a /= 4 means "divide a by 4" and update a to 6.0 (notice it becomes a float).

• a %= 4 means "get the remainder when a is divided by 4" and update a to 2.0.

Identity Operators

Check if two variables point to the same object:

o is Returns True if same object

o is not Returns True if not the same object

Syntax:

• a is b

True if a and b refer to the same object

• a is not b

True if a and b do NOT refer to the same object

Example program:

x = ["sprout", "coimbatore"]

y = ["sprout", "coimbatore"]

z = x

print(x is y): False

print(x is z): True

print(x is not y): True

Explanation:

- x and y have the same contents, but they are different objects in memory, so x is y is False.
- z is assigned to x, so both z and x point to the same object, making x is z True.
- Since x and y are not the same object, x is not y is True.

5. **Membership Operators**

Check if a value is present in a sequence (like a list or string): o in
Returns True if present

o not in Returns True if not present

Syntax:

➢ item in sequence

True if item is found inside the sequence

➢ item not in sequence

True if item is NOT found inside the sequence

Example program:

fruits = ["apple", "banana", "cherry"]

- print("banana" in fruits)

  Answer: True

- print("mango" in fruits)

Answer: False

- print("grape" not in fruits)

  Answer: True

Explanation:

- "banana" in fruits checks if "banana" is inside the list fruits. Since it is, the answer is True.
- "mango" in fruits asks if "mango" is in the list. It's not there, so the answer is False.
- "grape" not in fruits asks if "grape" is NOT in the list. Because it's not present, this is True.

AI Tutor Questions:

Operator Type	4C Skill	AI Tutor Question
Arithmetic	Critical Thinking	Why might dividing two integers sometimes give unexpected results in Python? Give an example.
	Creativity	Build a simple calculator that takes two numbers and performs all arithmetic operations.
	Communication	How would you explain the difference between * (multiply) and ** (power) to a new learner?

	Collaboration	Team up to write a calculator program. One writes the code, one tests edge cases. Swap roles.
Comparison	Critical Thinking	What would happen if you use = instead of == inside a comparison? Try and fix it.
	Creativity	Create a grading system that uses comparison operators to print messages like "Passed" or "Try Again" based on score.
	Communication	Explain why >= and <= might be more useful than just > or < in real-world situations like setting age limits.
	Collaboration	Pair up and build a student eligibility checker using marks and age. Share your logic and compare outputs.
Logical	Critical Thinking	When combining conditions with and/or, how does Python decide the final result? Give a case where logic changes the outcome.
	Creativity	Design a program to check if a user is eligible for an offer using multiple conditions like age, membership, and coupons.

	Communication	How would you explain the difference between and, or, and not to someone with no programming background?
	Collaboration	Work with a peer to create a quiz scoring system using logical operators to decide if the user passed or failed.
Assignment	Critical Thinking	What's the difference between x = x + 1 and x += 1? Why might you use one over the other?
	Creativity	Write a loop that updates a score variable using different assignment operators (+=, -=, *=).
	Communication	Explain the importance of using compound assignment operators (+=, -=) in writing cleaner code.
	Collaboration	One student writes a variable update scenario, and another modifies it using assignment shortcuts. Discuss the differences.
Identity	Critical Thinking	If two variables have the same value, are they always the same object in memory? Try with numbers and lists.
	Creativity	Create a quiz that shows

		id() values of different variables and asks users to guess if they're the same object.
	Communication	How would you teach the difference between is and == using everyday examples (e.g., identical twins vs same clothes)?
	Collaboration	Partner up to test different objects with is and ==. Share what you learned about Python memory behavior.
Membership	Critical Thinking	If "a" in "apple" is True, why is "A" in "apple" False? What does this say about membership checks?
	Creativity	Build a small word filter that checks if forbidden words exist in a message using in/not in.
	Communication	How would you explain the difference between in and not in when checking for values in a list or string?
	Collaboration	Create a login validator with a friend: one writes the username list, the other checks if input is in the list.

1.6. Control Flow

Control flow helps your program decide which code to run based on conditions. It's like making choices!

1.6.1. if-else Statement

Syntax

if condition:

code to run if condition is True else:

code to run if condition is False

Example program:

```
age = 18

if age >= 18:

print("You are eligible to vote.") else:

print("You eligible not to vote.")
```

Explanation:

- The program checks if age is 18 or more.
- If **True**, it prints "You are eligible to vote."
- If **False**, it prints "You eligible not to vote."

AI tutor question:

4C Skill	AI Tutor Question
Critical Thinking	Why do you think Python uses indentation instead of braces {} like other languages?
Creativity	Design a program that checks if a person is eligible for a game based on age and score.
Communication	How would you explain the concept of if-elif-else to a 10-year-old using a daily example?

Collaboration	With a partner, write a program that checks if today is a holiday, weekend, or working day.

1.6.2. Nested if Statement

You can put an if statement inside another if. This is called nested if.

Syntax:

if condition1:

if condition2:

//code if both conditions are True

Example program:

age = 20 is_registered

= True if age >= 18:

if is_registered:

print("You are eligible to vote.") else:

print("You need to register to vote.") else:

print("You are not eligible to vote yet.")

Explanation:

- First, it checks if the person is 18 or older (voting age).
- If yes, it then checks if they are registered to vote.

- If registered, it prints “You are eligible to vote.”
- If not registered, it reminds them to register.
- If under 18, it says they are not eligible yet.

AI Tutor Questions:

4C Skill	AI Tutor Question
Critical Thinking	What could go wrong if the inner if logic contradicts the outer condition?
Creativity	Design a game scenario where two conditions decide the next level.
Communication	Explain how nesting works with a real-world analogy, like school grades and age.
Collaboration	Partner 1 writes the outer if, Partner 2 writes a nested logic for eligibility.

1.6.3. Ternary Operator (Short if-else)

When you want a quick decision in one line, use the ternary operator.

Syntax:

value_if_true if condition else value_if_false

Example program:

```
age = 16

message = "Adult" if age >= 18 else "Minor"

print(message)
```

Explanation:

- It checks if age is 18 or more.

- If yes, message becomes "Adult".
- If no, message becomes "Minor".
- Then it prints the message.

AI Tutor Questions:

4C Skill	AI Tutor Question
Critical Thinking	When would a ternary operator be better than a full if- else?
Creativity	Create a ternary check that returns "Pass" or "Fail" based on marks.
Communication	Convert a ternary into normal if-else and explain both versions.
Collaboration	One writes the ternary version, the other writes the regular version. Compare.

1.7. Loops

Loops help you repeat a block of code again, either a specific number of times or until a condition is met.

while Loop

Use a while loop when you don’t know exactly how many times to repeat just keep going as long as the condition is True.

Syntax:

while condition:

//code to repeat

Example:

```
count = 1

while count <= 5:

 print("Count is:", count)

count += 1
```

Explanation:

- Start with count = 1.
- The loop checks: Is count less than or equal to 5?
- If yes, it prints the value and increases count by 1.
- This repeats until count becomes 6, then it stops.

AI Tutor Questions:

4C Skill	AI Tutor Question
Critical Thinking	What would happen if you forget to update the counter in a while loop? Try it.
Creativity	Make a pattern printer using nested loops (e.g., triangle of stars).
Communication	Explain to a beginner how break and continue affect loop flow with examples.
Collaboration	One writes a for loop that lists numbers, the other adds logic to skip odds.

for Loop

Use a for loop when you **know how many times** you want to repeat — often with a list or range of numbers.

Syntax:

```
for variable in sequence:
```

//code to repeat

Example program:

```
for i in range(1, 6):

print("Number:", i)
```

Explanation:

- range(1, 6) gives numbers from 1 to 5.
- Each time, i takes the next number.
- It prints i from 1 to 5.

AI Tutor Questions:

4C Skill	AI Tutor Question
Critical Thinking	What happens if your range is off-by-one? Can you fix an off-by-one error?
Creativity	Make a countdown from 10 to 1 using a for loop.
Communication	Teach a friend how range(start, stop) works with an example.
Collaboration	One writes a loop printing even numbers; the other handles odd numbers.

break Statement

break is used to stop a loop early when a condition is met.

Example:

```
for i in range(1, 10):

 if i == 5:

 break
```

print(i)

Explanation:

- This loop prints numbers from 1.
- But when i becomes 5, break stops the loop.
- Output: 1, 2, 3, 4

continue Statement

continue is used to skip the current loop step and move to the next one.

Example:

for i in range(1, 6):

if i == 3:

continue print(i)

Explanation:

- This prints numbers from 1 to 5.
- But when i is 3, it **skips** printing and goes to the next.
- Output: 1, 2, 4, 5 (3 is skipped)

AI tutor questions:

4C Skill	AI Tutor Question
Critical Thinking	What would happen if continue is placed before a critical print statement?
Creativity	Use continue to skip printing vowels from a word.
Communication	Explain the role of continue in controlling loop behavior.
Collaboration	One writes the loop, the other adds continue to handle a

	specific condition.

else with Loops

You can use else after a loop — it runs only if the loop completes normally (not stopped by break).

Example:

```
for i in range(1, 4):
 print("Checking:", i)
else:
 print("Loop completed.")
```

Explanation:

- It prints numbers 1 to 3.
- Then, since the loop finished without a break, the else runs. • Output ends with “Loop completed.”

AI tutor Questions:

4C Skill	AI Tutor Question
Critical Thinking	What happens if you add a break inside a loop with else?
Creativity	Build a number search loop with an else that says "Not found!"
Communication	Explain how loop else differs from if-else.

Collaboration	One writes a loop to find a number, the other adds else for post-loop message.

1.8. Functions

- Functions are like mini programs inside your program.
- You write the code once, give it a name, and then reuse it anytime by "calling" it.

def and return Syntax:

```
def function_name():
 # code inside the function
 return result
```

Example:

```
def greet():
 return "Hello, sprout"
message = greet()
print(message)
```

Explanation:

- def greet(): creates a function named greet.
- return "Hello, welcome!" sends back a message when the function is called.
- greet() is called and the result is stored in message.

• Then we print it.

AI tutor question:

4C Skill	AI Tutor Question
Critical Thinking	Why use functions instead of repeating the same code in multiple places? Give an example.
Creativity	Create a describe_pet function that prints info based on name and type of pet.
Communication	Teach someone how to define and call a function using your own simple example.
Collaboration	One writes a function to calculate squares of numbers, the other adds input logic and testing.

Function with Parameters

You can pass values (called parameters) into a function.

Syntax:

```
def greet(name):

print("Hello,", name)
```

Example:

```
def greet(name):

print("Hello,", name)

greet("sprout")
```

Explanation:

- The function expects a value called name.
- When you call greet("sprout"), it prints: Hello, sprout

4C AI Tutor Questions

4C Skill	AI Tutor Question
Critical Thinking	What happens if you forget to pass one of the required arguments to a function?
Creativity	Design a make_card(name, age) function that prints a greeting card message.
Communication	Explain the difference between a parameter and an argument to a peer.
Collaboration	One partner writes a function with two parameters, the other tests it with various values.

***args – Variable Number of Arguments** Use

*args to send a list of values to a function. **Syntax:**

```
def add_numbers(*args):
 # args behaves like a tuple
```

Example:

```
def add_numbers(*args):
 total = 0
 for num in args:
 total += num return
 total
```

print(add_numbers(10, 20, 30))

Explanation:

• *args allows the function to accept any number of values. • It adds 10 + 20 + 30 and returns 60.

****kwargs – Keyword Arguments**

Use **kwargs to pass values with names (keys).

Example:

```
def print_info(**kwargs):
 for key, value in kwargs.items():
 print(key, ":", value)
print_info(name="Sprout", age=6)
```

Explanation:

• **kwargs lets you pass data like a dictionary (key: value). • Here, name="Sprout" and age=6 are passed to the function. • The function prints each key-value pair.

AI Tutor Question:

- **Q1**: How does using *args help when you don't know how many arguments you'll need?
- **Follow-up**: Can you create a function that takes in any number of scores and returns their average?
- **Q2**: Collaborate on writing a flexible report generator using **kwargs. Try different inputs and discuss.

4C AI Tutor Questions

4C Skill	AI Tutor Question
Critical Thinking	Why would using *args or **kwargs be better than fixed parameters in some cases? Give an example.
Creativity	Create a log_event function that uses **kwargs to log events with keys like user, time, and action.

Communication	Explain to a friend the difference between *args and **kwargs using your own analogy.
Collaboration	One student writes a function using *args, another adds **kwargs and they combine use cases.

Lambda Functions (Short One-Line Functions)

Use lambda when you want a small, one-time function.
Syntax:

lambda arguments: expression

Example:

square = lambda x: x * x

print(square(5)) **Explanation:**

• lambda x: x * x creates a small function to square a number. • square(5) returns 25.

AI Tutor Question:

4C Skill	AI Tutor Question
Critical Thinking	Compare a regular function with a lambda function. When should you use each one?
Creativity	Create a lambda function that takes a number and returns "Even" or "Odd".
Communication	Teach your partner how to convert a normal function into a lambda function using a simple example.
Collaboration	One student writes a lambda to sort a list of names; the other provides test cases.

Variable Scope (Local and Global)

Variables inside functions are **local** — they can't be used outside.
Example:

```
def my_func():
 x = 10
 print("Inside:", x)
my_func()
print("Outside:", x) //This will cause an error
```

Explanation:

- x = 10 is created **inside** the function.
- When you try to use it **outside**, Python gives an error.
- It means variables have a **scope** (area of life).

AI tutor Question:

4C Skill	**AI Tutor Question**
Critical Thinking	Why might using too many global variables lead to errors in a big program? Give an example.
Creativity	Write a small game

	setup function where a local variable holds the player's name and prints it.
Communication	Explain to a friend how local and global variables differ. Use your own example.
Collaboration	One student writes a global variable; the other writes a function to modify it using global keyword.