Introduction to Python

What is Python?

Python is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991. Python emphasizes code readability and allows developers to express concepts in fewer lines of code compared to other languages like C++ or Java.

Key Features:

- **Interpreted**: Python code is executed line by line.
- **Dynamic Typing**: Variables do not need explicit declaration of their type.
- **Cross-Platform**: Python runs on Windows, macOS, Linux, and more.
- **Extensive Libraries**: Python has a rich ecosystem of libraries for tasks like data analysis, web development, and machine learning.

Applications:

- **Data Science and Machine Learning**: Libraries like Pandas, NumPy, Scikit-learn, and TensorFlow are widely used for data analysis, visualization, and building machine learning models.
- **Web Development**: Frameworks like Django and Flask.
- **Automation**: Automating repetitive tasks such as web scraping.
- **Game Development**: Libraries like Pygame allow developers to create 2D games using Python.

Python IDEs

An Integrated Development Environment (IDE) or text editor is essential for writing and running Python code efficiently.

- **Jupyter Notebook**
- **VS Code**
- **PyCharm**
- **Spyder**

Check the version of python in colab

In [1]:

```
!python --version
```

```
Python 3.11.11
```

Commenting and Uncommenting Code in Google Colab

- **Use the shortcut `Ctrl + /` to quickly comment or uncomment a selected line or block of code.**
- **This is a convenient way to toggle comments during debugging or testing without manually adding or removing comment symbols.**

Writing and Running Your First Python Program

In [2]:

```
print("hello world")
```

```
hello world
```

In [3]:

```
x = "Hello World"
print("Hello World")
print(x)
type(x)
```

```
Hello World
Hello World
```

Out[3]:

```
str
```

Note: The code cell below will not execute due to an error present in it.

In [ ]:

```
# print("a")
# print("b")
# print("c")
# print("d"
```

Defining variables

Variables are used to store data values.
Python uses dynamic typing, meaning you don't explicitly declare the type of a variable.

In [ ]:

```
name = "Alice"  # String variable
age = 30        # Integer variable
height = 5.8    # Float variable
is_student = True # Boolean variable
```

Variable Reassignment

- **The variable `a` is initially assigned the value `5`.**
- **It is then reassigned to the value `6`.**
- **The final value of `a` is `6`, as the last assignment overwrites the previous one.**

In [4]:

```
a = 5
a = 6

a
```

Out[4]:

```
6
```

Rules for naming variables

1. **Variable names can contain letters, numbers, and underscores.**
2. **Variable names cannot start with a number.**

3. Variable names are case-sensitive (age, Age, and AGE are different variables).
4. Avoid using reserved keywords (e.g., if, else, for, while, etc.).
5. Use descriptive names to make your code more readable.

In [ ]:

```
# Invalid variable names
# 1-variable = 10 # Invalid (starts with a number)
# my variable = 20 # Invalid (contains a space)
# if = 30        # Invalid (reserved keyword)
```

Avoid Overwriting Built-in Functions

- The line `# print = 2` is commented out to prevent execution.
- Assigning a value to a built-in function name like `print` (e.g., `print = 2`) is discouraged because it overrides the original functionality of the function, leading to potential errors and unexpected behavior.
- Always use unique variable names to avoid conflicts with Python's built-in functions.

In [ ]:

```
# Avoid using built-in function names for variables to prevent conflicts and errors.
# print = 2
```

In [ ]:

```
# Valid variable names
_my_variable = 100  # Valid (starts with underscore)
myVariable = 200    # Valid (camel case notation)
MyVariable = 300   # Valid (Pascal case notation)
```

In [ ]:

```
print(name)
print(age)
print(height)
print(is_student)
```

```
Alice
30
5.8
True
```

In [ ]:

```
# Modifying variable values
age = 31  # Change the value of the 'age' variable
print(age)
```

```
31
```

Data Types

Numeric

int (Integer):

Whole numbers

In [ ]:

```
age = 30
print(age, type(age))
```

```
30 <class 'int'>
```

Float

Numbers with decimal points

In [ ]:

```
# float (Floating-point):
height = 5.8
print(height, type(height))
```

```
5.8 <class 'float'>
```

String

Sequence of characters

In [ ]:

```
name = "danial"
```

Indexing

In [ ]:

```
print(name[2])
```

```
n
```

Slicing

In [ ]:

```
print(name[2:4])
```

```
ni
```

In [ ]:

```
print(name[:3])
```

```
dan
```

In [ ]:

```
print(name[2:])
```

```
nial
```

In [ ]:

```
# for persian
name = "دانیال"

name[0:4]
```

Out[ ]:

'دانی'

String methods

In [ ]:

```
text = "Hello, World!"
```

Upper & Lower

In [ ]:

```
uppercase_text = text.upper()
lowercase_text = text.lower()

print(f"Original: {text}")
print(f"Uppercase: {uppercase_text}")
print(f"Lowercase: {lowercase_text}")
```

```
Original: Hello, World!
Uppercase: HELLO, WORLD!
Lowercase: hello, world!
```

In [ ]:

```
name2 = 'ALi'
name2.lower()
```

Out[ ]:

```
'ali'
```

Replace

In [ ]:

```
new_text = text.replace("World", "Python")
print(f"Replaced: {new_text}")
```

```
Replaced: Hello, Python!
```

In [ ]:

```
text = "this is a book. the book is good"
text.replace("book", "pen")
```

Out[ ]:

```
'this is a pen. the pen is good'
```

In [ ]:

```
name = "ali"
name.replace('ali', 'reza')
```

Out[ ]:

```
'reza'
```

In [ ]:

```
# name[0] = 'l' -> Error, becuase string is immutable and we can't change it.
```

Capitalize

In [ ]:

```
city = "tehran"
city.capitalize()
```

Out[ ]:

```
'Tehran'
```

In [ ]:

```
city
```

Out[ ]:

```
'tehran'
```

In [ ]:

```
city = "tehran"
city2 = city.capitalize()
print(city)
print(city2)
```

```
tehran
Tehran
```

f-string

In [ ]:

```
name = "Alice"
age = 30
print(f"My name is {name} and I am {age} years old.")
```

```
My name is Alice and I am 30 years old.
```

In [5]:

```
fname = "sara"
lname = "mohammadi"
full_name = fname + " " + lname

age = 25

f"His name is {full_name} and he is {age} years old"
```

Out[5]:

```
'His name is sara mohammadi and he is 25 years old'
```

Concatenation

In [ ]:

```
"ali" + "reza"
```

Out[ ]:

```
'alireza'
```

List

Ordered, mutable (changeable) collection of items

In [ ]:

```
my_list = [1, 2, 3, "apple", True]
print(my_list, type(my_list))
```

```
[1, 2, 3, 'apple', True] <class 'list'>
```

Indexing

In [ ]:

```
my_list[2]
```

Out[ ]:

```
6
```

In [ ]:

```
my_list[-3]
```

Out[ ]:

```
4
```

Slicing

In [ ]:

```
my_list[1:3]
```

Out[ ]:

```
[5, 6]
```

List Methods

append

Adds an element to the end of the list

In [ ]:

```
my_list = [1, 2, 3, "apple", True]
```

In [ ]:

```
my_list.append(4)
print("After append:", my_list)
```

```
After append: [1, 2, 3, 'apple', True, 4]
```

insert

Inserts an element at a specific index

In [ ]:

```
my_list.insert(2, "banana")
print("After insert:", my_list)
```

```
After insert: [1, 2, 'banana', 3, 'apple', True, 4]
```

remove

Removes the first occurrence of a specific element

In [ ]:

```
my_list.remove("apple")
print("After remove:", my_list)
```

```
After remove: [1, 2, 'banana', 3, True, 4]
```

count

Returns the number of times a specific element appears in the list

In [ ]:

```
count_of_2 = my_list.count(2)
print("Count of 2:", count_of_2)
```

```
Count of 2: 1
```

pop

Removes and returns the element at a specific index (default is the last element)

In [ ]:

```
popped_element = my_list.pop(3)
print("Popped element:", popped_element)
print("After pop:", my_list)
```

```
Popped element: 3
After pop: [1, 2, 'banana', True, 4]
```

sort

Sorts the elements of the list in ascending order (in-place)

In [ ]:

```
numbers = [5, 2, 8, 1, 9]
numbers.sort()
print("Sorted Numbers:", numbers)
```

```
Sorted Numbers: [1, 2, 5, 8, 9]
```

Mutable vs Immutable

Key Concepts

- **Mutable Objects:** These are objects whose state or content can be modified after creation.
- **Immutable Objects:** These are objects whose state or content cannot be changed after creation.
- **Mutable Data Types:** Lists, Dictionaries, Sets
- **Immutable Data Types:** Integers, Floats, Strings, Tuple

Comparison: List (Mutable) vs String (Immutable)

List (Mutable)

Lists are mutable, meaning their elements can be modified after creation

In [6]:

```
# Example
  my_list = [1, 2, 3]
  my_list[0] = 10  # Modifying the first element
  print(my_list)
```

```
[10, 2, 3]
```

String (Immutable)

Strings are immutable, meaning their content cannot be changed after creation.

In [ ]:

```
my_string = "hello"
# my_string[0] = 'H'  # This would raise an error
```

In [ ]:

```
my_string.replace('h', 'H')
```

Out[ ]:

```
'Hello'
```

In [ ]:

```
my_string
```

Out[ ]:

```
'hello'
```

In [ ]:

```
new_string = my_string.replace('h', 'H')  # Creates a new string
print(new_string)
```

```
Hello
```

In [ ]:

```
my_list = [1, 2, 3, 'ali', True]

print(my_list, type(my_list))
```

```
[1, 2, 3, 'ali', True] <class 'list'>
```

In [ ]:

```
my_list[3] = 10
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 10, True]
```

In [ ]:

```
my_list[-1] = 20
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 10, 20]
```

In [ ]:

```
my_list.append(90)
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 10, 20, 90]
```

In [ ]:

```
my_list.insert(3, 4)
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 4, 10, 20, 90]
```

In [ ]:

```
my_list.extend([2, 4, 6])
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 4, 10, 20, 90, 2, 4, 6]
```

In [ ]:

```
my_list.append([2, 3, 4])
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 4, 10, 20, 90, 2, 4, 6, [2, 3, 4]]
```

In [ ]:

```
my_list[-1].append([1, 2, 3])
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 4, 10, 20, 90, 2, 4, 6, [2, 3, 4, [1, 2, 3]]]
```

In [ ]:

```
my_list[-1][1] = 5
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 4, 10, 20, 90, 2, 4, 6, [2, 5, 4, [1, 2, 3]]]
```

In [ ]:

```
my_list[-1][-1][0] = 10
```

In [ ]:

```
my_list
```

Out[ ]:

```
[1, 2, 3, 4, 10, 20, 90, 2, 4, 6, [2, 5, 4, [10, 2, 3]]]
```

In [ ]:

```
l1 = [1, 2, 3]
l2 = [4, 5, 6]

l3 = l1 + l2
```

In [ ]:

```
l3
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6]
```

In [ ]:

```
l1.extend(l2)
```

In [ ]:

```
l1
```

Out[ ]:

```
[1, 2, 3, 4, 5, 6]
```

In [ ]:

```
l1 = [1, 2, 3]
```

In [ ]:

```
l2 = l1
```

In [ ]:

```
print(id(l1), id(l2))
```

```
137370185386304 137370185386304
```

In [ ]:

```
print(l1)
```

```
print(l2)
```

```
[1, 2, 3]
[1, 2, 3]
```

In [ ]:

```
l2.append(4)
```

In [ ]:

```
print(l1)
print(l2)
```

```
[1, 2, 3, 4]
[1, 2, 3, 4]
```

In [ ]:

```
l1 = [1, 2, 3]
l2 = l1.copy()
```

In [ ]:

```
id(l1.copy())
```

Out[ ]:

```
137370186250816
```

In [ ]:

```
print(id(l1), id(l2))
```

```
137370186243968 137371031303808
```

In [ ]:

```
l2.append(4)
```

In [ ]:

```
print(l1)
print(l2)
```

```
[1, 2, 3]
[1, 2, 3, 4]
```

In [ ]:

```
lst = [1, 2, 3, 6, 7, 8]
lst[0]
```

Out[ ]:

```
1
```

In [ ]:

```
lst[1:4]
```

Out[ ]:

```
[2, 3, 6]
```

In [ ]:

```
lst[:100000]
```

Out[ ]:

```
[1, 2, 3, 6, 7, 8]
```

Tuple

Ordered, immutable (unchangeable) collection of items

In [ ]:

```
my_tuple = (1, 2, 3, "banana", False)
print(my_tuple, type(my_tuple))
```

```
(1, 2, 3, 'banana', False) <class 'tuple'>
```

Range

Represents a sequence of numbers

In [ ]:

```
my_range = range(5) # Numbers from 0 to 4
print(my_range, type(my_range))
```

```
range(0, 5) <class 'range'>
```

Dictionary

Collection of key-value pairs

In [8]:

```
my_dict = {
    "name": "Bob",
    "age": 25,
    "city": "New York"
}
print(type(my_dict))
```

```
<class 'dict'>
```

In [9]:

```
my_dict.get("name")
```

Out[9]:

```
'Bob'
```

In [10]:

```
my_dict.update({"age":26})
```

In [11]:

```
my_dict
```

Out[11]:

```
{'name': 'Bob', 'age': 26, 'city': 'New York'}
```

In [12]:

```
city = my_dict.pop("age")
```

In [13]:

```
print(city)
my_dict
```

```
26
```

Out[13]:

```
{'name': 'Bob', 'city': 'New York'}
```

In [14]:

```
my_dict.pop("city")
```

Out[14]:

```
'New York'
```

In [15]:

```
my_dict
```

Out[15]:

```
{'name': 'Bob'}
```

In [17]:

```
my_dict2 = {
    "name": "Bob",
    "age": 25,
    "city": "New York"
}
```

In [19]:

```
my_dict2.keys()
```

Out[19]:

```
dict_keys(['name', 'age', 'city'])
```

In [18]:

```
my_dict2.values()
```

Out[18]:

```
dict_values(['Bob', 25, 'New York'])
```

In [20]:

```
my_dict2.items()
```

Out[20]:

```
dict_items([('name', 'Bob'), ('age', 25), ('city', 'New York')])
```

Set

Unordered collection of unique items
Set items are unordered, unchangeable, and do not allow duplicate values.

In [29]:

```
my_set = {10, 50, 20, 3, 3, 4} # Duplicate 3 is removed
print(my_set, type(my_set))
```

```
{50, 3, 4, 20, 10} <class 'set'>
```

In [30]:

```
set1 = {"apple", "banana", "cherry"}
set2 = {"google", "microsoft", "apple"}
```

In [31]:

```
set3 = set1.union(set2)
set3
```

Out[31]:

```
{'apple', 'banana', 'cherry', 'google', 'microsoft'}
```

In [33]:

```
set4 = set1.intersection(set2)
set4
```

Out[33]:

```
{'apple'}
```

Boolean Type

Represents True or False

In [ ]:

```
is_student = True
print(is_student, type(is_student))
```

```
True <class 'bool'>
```

None

Represents the absence of a value

In [ ]:

```
result = None
print(result, type(result))
```

```
None <class 'NoneType'>
```

Type Casting (Conversion)

int to float

In [ ]:

```
age = 30
float_age = float(age)
print(f"Age as float: {float_age}, type: {type(float_age)}")
```

```
Age as float: 30.0, type: <class 'float'>
```

float to int

In [ ]:

```
height = 5.8
int_height = int(height)
```

```
print(f"Height as int: {int_height}, type: {type(int_height)}")
```

```
Height as int: 5, type: <class 'int'>
```

string to int

In [ ]:

```
string_num = "123"
int_num = int(string_num)
print(f"String num as int: {int_num}, type: {type(int_num)}")
```

```
String num as int: 123, type: <class 'int'>
```

int to string

In [ ]:

```
number = 42
string_number = str(number)
print(f"Number as string: {string_number}, type: {type(string_number)}")
```

```
Number as string: 42, type: <class 'str'>
```

string to float

In [ ]:

```
string_float = "3.14"
float_from_string = float(string_float)
print(f"String float as float: {float_from_string}, type: {type(float_from_string)}")
```

```
String float as float: 3.14, type: <class 'float'>
```

list to tuple

In [ ]:

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(f"List as tuple: {my_tuple}, type: {type(my_tuple)}")
```

```
List as tuple: (1, 2, 3), type: <class 'tuple'>
```

tuple to list

In [ ]:

```
my_tuple = (4, 5, 6)
my_list = list(my_tuple)
print(f"Tuple as list: {my_list}, type: {type(my_list)}")
```

```
Tuple as list: [4, 5, 6], type: <class 'list'>
```

input function

Get user input

In [ ]:

```
name = input("Enter your name: ")
age = int(input("Enter your age: "))  # Convert input to an integer
city = input("Enter your city: ")
```

```
Enter your name: name
Enter your age: 23
```

```
Enter your city: shz
```

In [ ]:

```
print("User Information:")
print("Name:", name)
print("Age:", age)
print("City:", city)
```

```
User Information:
Name: name
Age: 23
City: shz
```

Operators

Arithmetic Operators

In [ ]:

```
x = 10
y = 5

addition = x + y  # Addition
subtraction = x - y # Subtraction
multiplication = x * y # Multiplication
division = x / y # Division
floor_division = x // y # Floor division
modulus = x % y  # Modulus (remainder)
exponentiation = x ** y  # Exponentiation

print(f"Addition: {addition}")
print(f"Subtraction: {subtraction}")
print(f"Multiplication: {multiplication}")
print(f"Division: {division}")
print(f"Floor Division: {floor_division}")
print(f"Modulus: {modulus}")
print(f"Exponentiation: {exponentiation}")
```

```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
Floor Division: 2
Modulus: 0
Exponentiation: 100000
```

Comparison Operators

In [ ]:

```
a = 10
b = 5

print(f"Is a equal to b? {a == b}")  # Equal to
print(f"Is a not equal to b? {a != b}")  # Not equal to
print(f"Is a greater than b? {a > b}")  # Greater than
print(f"Is a less than b? {a < b}")  # Less than
print(f"Is a greater than or equal to b? {a >= b}")  # Greater than or equal to
```

```
print(f"Is a less than or equal to b? {a <= b}")  # Less than or equal to
```

```
Is a equal to b? False
Is a not equal to b? True
Is a greater than b? True
Is a less than b? False
Is a greater than or equal to b? True
Is a less than or equal to b? False
```

Comparison of Different Data Types

- **The expression `5 == '5'` compares an integer (`5`) with a string (`'5'`).**
- **In most programming languages, this comparison will return `False` because the data types (integer vs. string) are different, even though their values appear similar.**

In [ ]:

```
5 == '5'
```

Out[ ]:

```
False
```

Logical Operators

In [ ]:

```
p = True
q = False

print(f"p AND q: {p and q}")  # Logical AND
print(f"p OR q: {p or q}")  # Logical OR
print(f"NOT p: {not p}")  # Logical NOT
```

```
p AND q: False
p OR q: True
NOT p: False
```

Assignment Operators

In [ ]:

```
num1 = 15
num1 += 5 # num1 = num1 +5
print(f"num1 after += 5: {num1}")

num1 -= 3 # num1 = num1 - 3
print(f"num1 after -= 3: {num1}")

num1 *= 2 # num1 = num1 * 2
print(f"num1 after *= 2: {num1}")

num1 /= 4 # num1 = num1 / 4
print(f"num1 after /= 4: {num1}")

num1 %= 3 # num1 = num1 % 3
print(f"num1 after %= 3: {num1}")
```

```
num1 after += 5: 20
num1 after -= 3: 17
num1 after *= 2: 34
num1 after /= 4: 8.5
```

num1 after %= 3: 2.5

Conditional Statements

If statement

In [ ]:

```
x = 10
if x > 5:
  print("x is greater than 5")
```

```
x is greater than 5
```

If-else statement

In [ ]:

```
x = 9.9
if x >= 10:
  print("Passed")
else:
  print("Failed")
```

```
y is not greater than 5
```

In [ ]:

```
number = int(input("Enter a number: "))

if number % 2 == 0:
    print(f"{number} is even")
else:
    print(f"{number} is odd")
```

```
Enter a number: 2
2 is even
```

If-elif-else statement

In [ ]:

```
z = 7
if z > 10:
  print("z is greater than 10")
elif z > 5:
  print("z is greater than 5 but not greater than 10")
else:
  print("z is not greater than 5")
```

```
z is greater than 5 but not greater than 10
```

Nested if statements

In [ ]:

```
age = 20
if age >= 18:
  print("You are an adult")
  if age >= 65:
    print("You are also a senior citizen")
  else:
    print("You are not a senior citizen.")
else:
  print("You are a minor.")
```

```
You are an adult
You are not a senior citizen.
```

Using logical operators in conditions

In [ ]:

```
temperature = 25
if temperature > 20 and temperature < 30:
  print("The temperature is pleasant")
```

```
The temperature is pleasant
```

In [ ]:

```
is_raining = True
if not is_raining:
  print("Let's go outside!")
else:
  print("It's raining, stay inside!")
```

```
It's raining, stay inside!
```

Ternary operator

In [ ]:

```
result = "x is greater than 5" if x > 5 else "x is not greater than 5"
```

Loops

For loop

In [ ]:

```
for i in range(5):  # Iterate from 0 to 4
  print(i)
```

```
0
1
2
3
4
```

In [ ]:

```
names = ['ali', 'reza', 'mina', 'sara']
for name in names:
  print(f"hello {name}")
```

```
hello ali
hello reza
hello mina
hello sara
```

In [ ]:

```
names = ['ali', 'reza', 'mina', 'sara']
for i in range(len(names)):
  print(f"student {i}: {names[i]}")
```

```
student 0: ali
student 1: reza
student 2: mina
student 3: sara
```

In [ ]:

```
fact = 1
n = int(input("factorial of n: "))
for i in range(1, n+1):
  fact *= i
  print(fact)

print(f"factorial of {n} is {fact}")
```

```
factorial of n: 5
1
2
6
24
120
factorial of 5 is 120
```

Break statement

In [ ]:

```
for i in range(10):
  if i == 5:
    break  # Exit the loop when i is 5
  print(i)
```

```
0
1
2
3
4
```

Continue statement

In [35]:

```
for i in range(10):
  if i % 2 == 0:
    continue  # Skip even numbers
  print(i)
```

```
1
```

```
1
3
5
7
9
```

While loop

In [ ]:

```
d = 5
fact = 1
c = 1
while c <= d:
  fact = fact * c
  c += 1
print(fact)
```

```
120
```

In [ ]:

```
user_input = input("Enter a number (or 'q' to quit): ")
while user_input != 'q':
  print("Still There!")
  user_input = input("Enter a number (or 'q' to quit): ")
```

```
Enter a number (or 'q' to quit): q
```

In [ ]:

```
while True:
  user_input = input("Enter a number (or 'q' to quit): ")
  if user_input == 'q':
    break
  print("Still There!")
```

```
Enter a number (or 'q' to quit): q
```

Function

Define a function

In [ ]:

```
b = 6 # global variable
def add_numbers():
  c = 5 # local variable
  return a + b
```

In [ ]:

```
print(a)
```

```
10
```

In [ ]:

```
print(c) # we got an error, because c is a local variable
```

```
---------------------------------------------------------------------------
```

```
NameError                                 Traceback (most recent call last)
<ipython-input-138-1dd5973cae19> in <cell line: 0>()
----> 1 print(c)

NameError: name 'c' is not defined
```

In [ ]:

```
result = add_numbers()
print(result)
```

```
11
```

Positional Argument

In [ ]:

```
def add_numbers2(a, b):
  return a + b
```

In [ ]:

```
result = add_numbers2(2, 3)
print(result)
```

```
5
```

Keyword Argument

In [ ]:

```
def greet(name, greeting):
  return f"{greeting}, {name}!"

result = greet(greeting="Hi", name="Alice")
print(result)
```

```
Hi, Alice!
```

Default argument

In [ ]:

```
def add_numbers3(a, b=5):
  return a + b
```

In [ ]:

```
result = add_numbers3(3)
print(result)
```

```
8
```

In [ ]:

```
def factorial(n):
  fact = 1
  for i in range(1, n+1):
    fact *= i
```

```
  return fact

n = int(input("factorial of n: "))
fact = factorial(n)
print(fact)
```

```
factorial of n: 3
6
```

In [ ]:

```
def fibo(i):
  f1 = 0
  f2 = 1
  if i<= 2:
    return i-1
  for c in range(3,i+1):
    f3 = f2 + f1
    f1 = f2
    f2 = f3
  return f3

for x in range(1,8):
  print(fibo(x))
```

```
0
1
1
2
3
5
8
```

Exception Handling

In [ ]:

```
number = int(input("Enter a number: "))
```

```
Enter a number: s

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-123-76c9e59d7c5b> in <cell line: 0>()
----> 1 number = int(input("Enter a number: "))

ValueError: invalid literal for int() with base 10: 's'
```

In [ ]:

```
try:
  # Code that might raise an exception
  number = int(input("Enter a number: "))
  print(number)
except Exception as e:
  # Handle the specific exception
  print(f"Error: {e}")
```

```
Enter a number: d
Error: invalid literal for int() with base 10: 'd'
```

In [ ]:

```
10 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-125-cd759d3fcf39> in <cell line: 0>()
```

```
----> 1 10 / 0

ZeroDivisionError: division by zero
```

In [ ]:

```python
try:
  result = 10 / 0
except ZeroDivisionError as e:
  print(f"Error: {e}")
```

```
Error: division by zero
```