Python, like many other programming languages, utilizes various **data types** to categorize and manage different kinds of information. Each data type specifies the type of values a variable can hold and the operations that can be performed on those values. Here's an overview of the common data types in Python:

**1. Numeric Types:**

* **int (integer):** Represents whole numbers (positive, negative, or zero) with unlimited precision. Examples: 10, -5, 0
* **float (floating-point):** Represents real numbers with a decimal point, typically offering limited precision (around 15 decimal places). Examples: 3.14, -9.87, 1.0e3 (scientific notation for 1000)
* **complex:** Represents complex numbers, containing a real and an imaginary part separated by "j". Examples: 3+5j, 1.2j, -2-4.5j

**2. String Type:**

* **str (string):** Represents sequences of characters, including letters, numbers, symbols, and whitespace. Enclosed in single (') or double (") quotes. Examples: "Hello, world!", 'This is a string', '123'

**3. Sequence Types:**

* **list:** Represents an ordered, mutable collection of elements enclosed in square brackets []. Elements can be of different data types. Examples: [1, 2, 3, "apple"], [3.14, True, None], [] (empty list)
* **tuple:** Represents an ordered, immutable collection of elements enclosed in parentheses (). Elements can be of different data types. Examples: (1, "cat", 4.2), ("hello",) (single-element tuple), () (empty tuple)

**4. Set Type:**

* **set:** Represents an unordered collection of unique elements enclosed in curly braces {}. Elements can be of different data types. Examples: {1, 2, 3, "apple"}, {"hello", 3.14}, set() (empty set)

**5. Dictionary Type:**

* **dict (dictionary):** Represents an unordered collection of key-value pairs enclosed in curly braces {}. Keys must be unique and immutable (often strings or numbers). Values can be of any data type. Examples: {"name": "Alice", "age": 30, "city": "New York"}, {"x": 1, "y": 2.5}, {} (empty dictionary)

**6. Boolean Type:**

* **bool:** Represents logical values, either True or False. Used for conditional statements and boolean expressions. Examples: True, False

**7. None Type:**

* **None:** Represents the absence of a value. Often used as a placeholder or to indicate that a variable is not associated with any specific data.

Understanding data types is crucial for effectively working with data in Python programs. By choosing the appropriate data type for your variables, you ensure data integrity, improve code readability, and prevent potential errors.

Python, like most programming languages, utilizes various **operators** to perform operations on values and variables. These operators allow you to manipulate data, control the flow of your program, and perform calculations. Here's a breakdown of the common operators in Python:

**1. Arithmetic Operators:**

* **+ (addition):** Adds two operands. Examples: 5 + 3 = 8, 1.2 + 4.5 = 5.7
* **- (subtraction):** Subtracts the second operand from the first operand. Examples: 10 - 2 = 8, 3.14 - 1.5 = 1.64
* **\* (multiplication):** Multiplies two operands. Examples: 4 \* 5 = 20, 2.5 \* 3 = 7.5
* \*\* / (division):\*\* Divides the first operand by the second operand. In Python 3, it always returns a float, even for integer division. Examples: 10 / 2 = 5.0, 7 / 2 = 3.5
* \*\* // (floor division):\*\* Divides the first operand by the second operand and returns the integer quotient, discarding any remainder. Examples: 10 // 3 = 3, 7 // 2 = 3
* \*\* % (modulo):\*\* Returns the remainder of dividing the first operand by the second operand. Examples: 10 % 3 = 1, 7 % 2 = 1
* \*\* \*\* (exponentiation):\*\* Raises the first operand to the power of the second operand. Examples: 2 \*\* 3 = 8, 1.5 \*\* 2 = 2.25

**2. Comparison Operators:**

* **== (equal to):** Checks if two operands are equal. Examples: 5 == 5 is True, 10 == 2 is False
* **!= (not equal to):** Checks if two operands are not equal. Examples: 3 != 3 is False, 7 != 2 is True
* **> (greater than):** Checks if the first operand is greater than the second operand. Examples: 8 > 5 is True, 2 > 7 is False
* **< (less than):** Checks if the first operand is less than the second operand. Examples: 3 < 5 is True, 7 < 2 is False
* **>= (greater than or equal to):** Checks if the first operand is greater than or equal to the second operand. Examples: 10 >= 5 is True, 3 >= 3 is True
* **<= (less than or equal to):** Checks if the first operand is less than or equal to the second operand. Examples: 2 <= 5 is True, 7 <= 7 is True

**3. Logical Operators:**

* **and:** Returns True if both operands are True, otherwise False. Examples: True and True is True, False and True is False
* **or:** Returns True if at least one operand is True, otherwise False. Examples: True or False is True, False or False is False
* **not:** Inverts the truth value of the operand. Examples: not True is False, not False is True

**4. Assignment Operators:**

* **= (assignment):** Assigns the value of the right operand to the left operand. Examples: x = 10, y = "hello"
* **+=, -=, \*=, /=, //=, %=** (combined assignment operators): Combine the assignment operation with the corresponding arithmetic operation. Examples: x += 5 (equivalent to x = x + 5)

**5. Identity Operators:**

* **is:** Checks if two operands are the same object in memory. Examples: x is y is True if x and y refer to the same object
* **is not:** Checks if two operands are not the same object in memory. Examples: x is not y is True if x and y refer to different objects

**6. Membership Operators:**

* **in:** Checks if a value is present in a sequence (list, tuple, string). Examples: 3 in [1, 2, 3] is True, "world" in "Hello, world!" is True
* **not in:** Checks if a value is not present in a sequence. Examples: 5 not in [1, 2, 3] is True, "cat" not in "dog" is True

**7. Bitwise Operators (applicable to integers):**

* **& (bitwise AND):** Performs

In Python, **operator precedence** determines the order in which operations are evaluated within an expression. This is crucial because it affects the final result of your calculations. Here's a breakdown of operator precedence in Python:

**1. Parentheses** ()**:** Parentheses have the highest precedence. Any expression within parentheses is evaluated first, regardless of other operators present.

**2. Exponentiation** \*\***:** Exponentiation (raising a number to a power) comes next.

**3. Multiplication** \* **and Division** /**:** Multiplication and division have the same precedence and are evaluated from left to right.

**4. Addition** + **and Subtraction** -**:** Addition and subtraction have the same precedence and are also evaluated from left to right.

**5. Comparison Operators:** Comparison operators (like ==, !=, >, <, >=, <=) have lower precedence than arithmetic operators. They are evaluated from left to right.

**6. Logical Operators:** Logical operators (and, or, not) have even lower precedence than comparison operators. They are also evaluated from left to right.

Here are some examples to illustrate:

* 2 + 3 \* 4 will be evaluated as 2 + (3 \* 4) = 14 (multiplication before addition due to same precedence and left-to-right evaluation)
* (2 + 3) \* 4 will be evaluated as (5 \* 4) = 20 (parentheses have higher precedence)
* 5 > 2 and 3 < 1 will be evaluated as False (comparison operators evaluated from left to right, 5 > 2 is True, but 3 < 1 is False, and and requires both conditions to be True)

Remember, **use parentheses whenever necessary to clarify the order of operations** and avoid unexpected results. You can also use a tool called an **operator precedence table** to easily reference the order of different operators.

**1. Single-line comments:**

* Start with a hash symbol (#) and extend to the end of the line.
* Used for brief explanations or comments related to the specific line of code they follow.

**Example:**

Python

# This variable stores the user's name

name = input("Enter your name: ")

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**2. Multi-line comments:**

* Enclosed within triple quotation marks (""" or '''''') on separate lines.
* Useful for providing detailed explanations, descriptions of functions or algorithms, or documenting code sections.

**Example:**

Python

"""

This function calculates the area of a circle.

Args:

radius: The radius of the circle (float).

Returns:

The area of the circle (float).

"""

def circle\_area(radius):

return math.pi \* radius \*\* 2

Use code [with caution.](https://gemini.google.com/faq#coding)

content\_copy

**Best practices for using comments:**

* **Clarity:** Strive for clear, concise, and accurate comments that explain the code's functionality.
* **Placement:** Place comments strategically to improve the overall readability of the code.
* **Avoid redundancy:** Don't repeat what the code already says. Instead, explain the "why" behind the code's implementation.
* **Document assumptions:** Comment on any assumptions made within the code or constraints that apply.

Remember, commenting is an essential aspect of writing good quality Python code. By effectively using comments, you can significantly enhance your code's maintainability and understanding for yourself and others.

import math

print("Hello, Janita Phatak here!")

print("How do you do?")

print("Variables are aprototype of container which are used to store the data")

x=10

y="janita"

z=math.pi

print(x,y,z)

print("the variable name can have numbers but not at 1st place , invalid--2x, valid--a2")

print("you cant put special character and spaces in variable except '\_ '")

x\_=5

print(x\_)

j=9

h=0.5

k="100"

print(type(j))

adult=True

married=False

print(type(adult))

a=2

b=3

c=a+b+2

d=8+3

print(c,d)

#calculating area

# import math

# radius = float(input("Enter the radius of the circle: "))

# area = math.pi \* radius \*\* 2

# print(f"The area of the circle is: {area:.2f}")

"""multiple comment"""

'''''multiple comment'''''

name="janita"

print("hi "+ name)

print("hiiiiiiiiii",name)

l=0.888888

print(f"{l:.3f}")

f\_number = int(input("Enter first number: "))

s\_number = int(input("Enter second number: "))

operation = input("Enter desired operation (+ or - or \* or / ) : ")

def calculator(f\_number, s\_number,operation):

if operation not in "+-/\*":

print("Invalid Operation Symbol")

return None

if operation == "+":

result = f\_number + s\_number

elif operation =="-":

result = f\_number - s\_number

elif operation == "\*":

result = f\_number \* s\_number

else:

result = f\_number / s\_number

return result

result = calculator(f\_number,s\_number,operation)

if result is not None:

print (f"The result of {f\_number} {operation} {s\_number} is : {result} ")

#Calculating BMI

# weight = float(input("Enter your weight in kilograms : "))

# height = float(input("Enter your height in meters : "))

# BMI = weight / ( height \* height )

# print(f"BMI is: {BMI:.2f}" )

#Making Conversion

# temperature = float(input("Enter temperature in celcius : "))

# fahrenheit = (temperature \* 9/5) + 32

# print(temperature ,"celicus equals to", fahrenheit , "fahrenheit")

def convert\_temperature(temperature, from\_unit, to\_unit):

"""Converts temperature from one unit to another."""

# Define conversion factors and formulas

conversion\_factors = {

"C\_to\_F": lambda c: (c \* 9/5) + 32,

"F\_to\_C": lambda f: (f - 32) \* 5/9,

# Add more conversions as needed (e.g., C\_to\_K, F\_to\_K)

}

# Validate input units

if from\_unit not in conversion\_factors or to\_unit not in conversion\_factors:

raise ValueError("Invalid temperature unit(s). Please use 'C' for Celsius or 'F' for Fahrenheit.")

# Get the conversion function based on units

conversion\_function = conversion\_factors[f"{from\_unit}\_to\_{to\_unit}"]

# Convert the temperature

converted\_temperature = conversion\_function(temperature)

# Return the converted temperature with the unit symbol

return f"{converted\_temperature:.2f} {to\_unit}"

# Get user input

temperature = float(input("Enter the temperature: "))

from\_unit = input("Enter the unit (C or F): ").upper()

to\_unit = input("Enter the desired unit (C or F): ").upper()

# Try-except block to handle potential errors

try:

converted\_temperature = convert\_temperature(temperature, from\_unit, to\_unit)

print(f"The converted temperature is: {converted\_temperature}")

except ValueError as e:

print(e) # Print the error message