Mutable and Immutable : which cant be changed

```
# Mutable and Immutable



rent = "5000"

rent = 300

print(rent)

# String is not mutable

my_string = "Anish"

# my_string[0]= 'M'

my_string = "M" + my_string.lower()

print(my_string

)
```

#WAP to split your first_name and last_name

```


my_full_name = "Anish Manandhar"

print(my_full_name.split(' '))
```

Usage of \

```
# Print "Let's learn Python", said Jenish

print('"let\'s learn python", said Jenish')
```

type

`print(type(rent))

Data Types

int , float , str , boolean,None, complex, list, dictionary,range

Input

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

print(type(my_number))
```

Variable Name Declaration

Identifier Type	Naming Style	Example
Variables & Functions	`snake_case` (lowercase with underscores)	`user_name`, `calculate_total()`
Constants	`UPPER_CASE_WITH_UNDERSCORES`	`MAX_LIMIT`, `PI`
Classes	`PascalCase` (or CapWords)	`UserProfile`, `DatabaseConnection`
Modules	Short, all-lowercase, underscores if needed	`math_utils.py`
Unused Variables	Single underscore	`for _ in range(10):`

Operators

```
- Arithmetic, Assignment, Comparision, Logical, Membership, Bitwise
```

In Python, **operators** are symbols used to perform operations on values and variables.

1. Arithmetic operators

Used for math.

```
a = 10
b = 3
```

```
print(a + b)   # 13  addition
print(a - b)   # 7   subtraction
print(a * b)   # 30  multiplication
print(a / b)   # 3.333... division
print(a // b)  # 3   floor division
print(a % b)   # 1   modulus / remainder
print(a ** b)  # 1000 exponent / power
```

2. Assignment operators

Used to assign or update values.

```
x = 5
x += 2   # same as x = x + 2
x -= 1   # same as x = x - 1
x *= 3   # same as x = x * 3
x /= 2   # same as x = x / 2
x //= 2
x %= 2
x **= 3
```

3. Comparison operators

Used to compare values. They return `True` or `False`.

```
a = 10
b = 5

print(a == b)  # False, equal to
print(a != b)  # True, not equal to
print(a > b)   # True, greater than
print(a < b)   # False, less than
print(a >= b)  # True, greater than or equal to
print(a <= b)  # False, less than or equal to
```

4. Logical operators

Used to combine conditions.

```
age = 20

print(age > 18 and age < 30)  # True
```

```
print(age < 18 or age > 60)   # False
print(not age > 18)           # False
```

5. Membership operators

Used to check whether a value exists inside something.

```
name = "Anish"

print("A" in name)      # True
print("z" not in name)  # True
```

6. Identity operators

Used to check whether two variables refer to the same object.

```
a = [1, 2]
b = a
c = [1, 2]

print(a is b)      # True
print(a is c)      # False
print(a == c)      # True
print(a is not c)  # True
```

`==` checks value.
`is` checks object identity.

7. Bitwise operators

Used to work with binary numbers.

```
a = 5   # binary: 0101
b = 3   # binary: 0011

print(a & b)   # 1  AND
print(a | b)   # 7  OR
print(a ^ b)   # 6  XOR
print(~a)      # -6 NOT
print(a << 1)  # 10 left shift
print(a >> 1)  # 2  right shift
```

#WAP *WAP to find a number is even or odd*

```
my_number = 0

if my_number == 0:

print("Zero")

elif my_number %2 ==0:

print("Even")

else:

print("Odd")
```

```
n = 7

if n & 1:
    print("Odd")
else:
    print("Even")
```

```
Useful in Booth's multiplication algorithm ,Binary Division , CRC, Parity
Checking, Decoding, ALU , Cache Design
```

List

A **list** stores multiple values in order. It is **mutable**, meaning you can change it.

```
item_list =['rent','lunch','dinner']

item_list[0] = 'breakfast'



print(item_list)

item_list.append('travel')

print(item_list)

item_list.insert(3,'trek')
```

```
print(item_list)

item_list.pop()

print(item_list)
```

Looping in List

```
for item in item_list:
     print(item)
```

Membership Operator in List

`print('trek' in item_list)`

Important

List Comprehension

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

squared_list = []

for number in number_list:

squared_list.append(number**2)

print(squared_list)
```

can be transposed to

```
squared_list_comprehension = [number**2 for number in number_list ]

print(squared_list_comprehension)


squared_odd_list_comprehension = [number**2 for number in number_list if
```

```
number &1 ]

print(squared_odd_list_comprehension)
```

Simple formula
`[what_to_store for item in collection if condition]`

Nested List

```
#nested list

nested_list = [1,2,[3,4],[5,[100,200,['hello']],23,11],1,7]

print(nested_list[3])
```

- List can be hetereogeneous

Tuples

A **tuple** is like a list, but it is **immutable**, meaning you cannot change it after creating it.

```
#Tuple have the heterogeneous values eg x-coordinate and y coordinate

point =(4,5)

point
```

```
Try setting 'point[0]=2'
```

Sets

A **set** stores unique values only. It is **mutable**, but it does **not keep duplicate values**.

```
a = {1, 2, 3,3,2}

b = {3, 4, 5}

print(a)

print(a | b) # union: {1, 2, 3, 4, 5}

print(a & b) # intersection: {3}
```

```
print(a - b) # difference: {1, 2}
```

Dictionary

```
subject_dictionary = {

"Microprocessor": "Gajendra Sharma",

"MCSC": "Gokul",

"DSA": "Deni",

"DBMS": "Rajani",

}

print(subject_dictionary)

print(subject_dictionary["Microprocessor"])

subject_dictionary['MCSC'] = 'Shrayan'

print(subject_dictionary)
```

Looping in dictionary

```


for key in subject_dictionary:

print(f'{key} is taught by {subject_dictionary[key]}')
```

Key Value and Pairs

```
print(subject_dictionary.items())

print(subject_dictionary.keys())

print(subject_dictionary.values())
```

Pop also works similarly

```
subject_dictionary.pop('MCSC')

print(subject_dictionary.keys())
```

Type	Syntax	Ordered?	Mutable?	Allows duplicates?	Example use
List	`[]`	Yes	Yes	Yes	Many items
Tuple	`()`	Yes	No	Yes	Fixed values
Set	`{}`	No reliable index order	Yes	No	Unique items
Dictionary	`{key: value}`	Yes	Yes	Keys unique	Labeled data

Loop

While

```
i=1

while i<=5 :

print(i)

i=i+1
```

range(start,end)

`print(list(range(0,10,2)))`

#WAP to find squares from 1,3,5,7...21 in list form

```
squares = [n ** 2 for n in range(1, 22) if n&1]
print(squares)
```

Functions

```
def greet(name="Guest"):
    print("Hello", name)
```

```
greet("Anish")
#default value
greet()


#Function with return value
def add(a, b):
    return a + b

result = add(5, 3)
print(result)

#Function with multiple parameters with keyword arguments

def student(name, course):
    print(name, "is learning", course)

student(course="Python", name="Anish")
```

Documentation String

```
print(help(range))
print(range.__doc__)
```

```
def add(a, b):
    """
    Return the sum of two numbers.    """    return a + b

print(help(add))
```

#WAP to find area of triangle, rectangle with based and height

```
def calculate_area(dimension1,dimension2,shape="triangle"):

''' documentation string

:param dimension1: In case of triangle it is "base". For rectangle it is
"length".

:param dimension2: In case of triangle it is "height". For rectangle it is
"width".

:param shape: Either "triangle" or "rectangle"
```

```
:return: Area of a shape

'''

if shape=="triangle":

area=1/2*(dimension1*dimension2) # Triangle area is : 1/2(Base*Height)

elif shape=="rectangle":

area=dimension1*dimension2 # Rectangle area is: Length*Width

else:

print("***Error: Input shape is neither triangle nor rectangle.")

area=None # If user didn't supply "triangle" or "rectangle" as shape then
return None

return area



# Calculate area of triangle whose base is 10 and height is 5

base=10

height=5

triangle_area=calculate_area(base,height,"triangle")

print("Area of triangle is:",triangle_area)
```

Args & Kwargs

```
# If the arguments to be be passed are unknown
def last_participant(instructor,*participant):

#     print(type(participant))
    print("The instructor is ",instructor)
    print("The last participant is " + participant[-1])

last_participant("Anish","Suraj", "Sushan", "Samman","SHyam")
```

```
# If the arguments to be be passed are unknown
def last_participant(instructor,*participant,**heros):

#     print(type(participant))
    print("The instructor is ",instructor)
    print("The last participant is " + participant[-1])

    print("\n Now meet the heros behind the workshop")
    print(f"{heros['club']},{heros['coordinator']}")


last_participant("Anish","Suraj", "Sushan", "Samman",coordinator
='Adarsha',club='Swastik IT CLUB')
```

Lambda Function

```
#Lambda Function
def square_of_number(n):
    return n**2

print(square_of_number(3))
t = lambda x : x**2
print(t(6))
```