**✅ Covered Concepts:**

* Type Casting
* 5 Operators:
  + Arithmetic
  + Comparison
  + Assignment
  + Logical
  + Membership
* Identity Operators
* Ternary Operator
* Operator Precedence
* is vs ==

**🧺 1. LIST**

# ✅ Type Casting

my\_tuple = (1, 2, 3)

my\_list = list(my\_tuple)

# ✅ Arithmetic Operators (only on elements, not the list directly)

nums = [1, 2, 3]

result = [x \* 2 for x in nums] # ➝ [2, 4, 6]

# ✅ Comparison Operators

a = [1, 2]

b = [1, 2]

print(a == b) # ✅ True (same content)

print(a is b) # ❌ False (different memory locations)

# ✅ Assignment Operators

a = [1, 2]

a += [3] # ➝ [1, 2, 3]

# ✅ Logical Operators

a = [1]

b = []

print(bool(a and b)) # False

# ✅ Membership

print(2 in [1, 2, 3]) # True

# ✅ Identity

x = [1, 2]

y = x

print(x is y) # True (same object)

# ✅ Ternary

x = [1, 2]

msg = "Non-empty" if x else "Empty"

# ✅ Operator Precedence

# Example: not x in y → 'not' has lower precedence than 'in'

nums = [1, 2, 3]

print(not 4 in nums) # True

# ✅ is vs ==

print([1, 2] == [1, 2]) # True

print([1, 2] is [1, 2]) # False

**📦 2. TUPLE**

# ✅ Type Casting

my\_list = [1, 2, 3]

my\_tuple = tuple(my\_list)

# ✅ Arithmetic (repetition only)

t = (1, 2)

print(t \* 2) # ➝ (1, 2, 1, 2)

# ✅ Comparison

print((1, 2) == (1, 2)) # True

# ✅ Assignment

t = (1, 2)

t += (3,) # ➝ (1, 2, 3)

# ✅ Logical

print(bool(()) and True) # False

# ✅ Membership

print(2 in (1, 2, 3)) # True

# ✅ Identity

a = (1, 2)

b = (1, 2)

print(a is b) # False (might be True for small ints)

# ✅ Ternary

msg = "OK" if (1,) else "Empty"

# ✅ Precedence

print(not 5 in (1, 2, 3)) # True

# ✅ is vs ==

print((1, 2) == (1, 2)) # True

print((1, 2) is (1, 2)) # False

**📚 3. DICTIONARY**

# ✅ Type Casting

pairs = [("a", 1), ("b", 2)]

d = dict(pairs)

# ✅ Arithmetic (not allowed on dicts directly)

# ✅ Comparison

d1 = {"a": 1}

d2 = {"a": 1}

print(d1 == d2) # True

# ✅ Assignment

d = {"a": 1}

d["b"] = 2 # ➝ {"a": 1, "b": 2}

# ✅ Logical

print(bool({}) or True) # True

# ✅ Membership (checks \*\*keys\*\* only)

print("a" in {"a": 1, "b": 2}) # True

# ✅ Identity

x = {"a": 1}

y = x

print(x is y) # True

# ✅ Ternary

d = {"val": 10}

result = "Present" if "val" in d else "Missing"

# ✅ Precedence

# Membership has higher precedence than `not`

print(not "x" in {"a": 1}) # True

# ✅ is vs ==

print({"a": 1} == {"a": 1}) # True

print({"a": 1} is {"a": 1}) # False

**🔘 4. SET**

# ✅ Type Casting

my\_list = [1, 2, 2, 3]

my\_set = set(my\_list) # ➝ {1, 2, 3}

# ✅ Arithmetic (Union/Intersection etc.)

a = {1, 2}

b = {2, 3}

print(a | b) # ➝ {1, 2, 3}

print(a & b) # ➝ {2}

# ✅ Comparison

print({1, 2} == {2, 1}) # True (unordered)

# ✅ Assignment

a = {1}

a.add(2)

# ✅ Logical

print(bool(set()) or False) # False

# ✅ Membership

print(2 in {1, 2, 3}) # True

# ✅ Identity

x = {1, 2}

y = x

print(x is y) # True

# ✅ Ternary

s = {1}

msg = "Filled" if s else "Empty"

# ✅ Precedence

print(not 5 in {1, 2}) # True

# ✅ is vs ==

print({1, 2} == {2, 1}) # True

print({1, 2} is {1, 2}) # False

**🧊 5. FROZENSET**

# ✅ Type Casting

s = frozenset([1, 2, 2, 3]) # ➝ frozenset({1, 2, 3})

# ✅ Arithmetic (Set operations)

f1 = frozenset([1, 2])

f2 = frozenset([2, 3])

print(f1 | f2) # ➝ frozenset({1, 2, 3})

# ✅ Comparison

print(frozenset([1, 2]) == frozenset([2, 1])) # True

# ✅ Assignment (immutable – no add/remove)

# ✅ Logical

print(bool(frozenset()) and True) # False

# ✅ Membership

print(2 in frozenset([1, 2])) # True

# ✅ Identity

x = frozenset([1])

y = x

print(x is y) # True

# ✅ Ternary

fs = frozenset()

msg = "OK" if fs else "Empty"

# ✅ Precedence

print(not 9 in frozenset([1, 2])) # True

# ✅ is vs ==

print(frozenset([1, 2]) == frozenset([2, 1])) # True

print(frozenset([1, 2]) is frozenset([1, 2])) # False

**📌 Summary Table**

| **Concept** | **✅ Supported In** | **🛑 Not Applicable To** |
| --- | --- | --- |
| Type Casting | All | — |
| Arithmetic Ops | Set, Tuple (repeat), List (via loops) | Dict, Frozenset (direct ops) |
| Comparison Ops | All | — |
| Assignment Ops | All (except frozen) | Frozenset (immutable) |
| Logical Ops | All | — |
| Membership Ops | All | — |
| Identity (is) | All | — |
| Ternary | All | — |
| Precedence Rules | All | — |
| is vs == | All | — |

# Python Data Structures and Operators Guide

## Type Casting

Type casting allows you to convert between different data types.

# List casting

list\_from\_tuple = list((1, 2, 3)) # (1, 2, 3) → [1, 2, 3]

list\_from\_set = list({4, 5, 6}) # {4, 5, 6} → [4, 5, 6]

list\_from\_dict = list({'a': 1, 'b': 2}) # {'a':1, 'b':2} → ['a', 'b']

# Tuple casting

tuple\_from\_list = tuple([1, 2, 3]) # [1, 2, 3] → (1, 2, 3)

tuple\_from\_set = tuple({4, 5, 6}) # {4, 5, 6} → (4, 5, 6)

tuple\_from\_dict = tuple({'a': 1, 'b': 2}) # {'a':1, 'b':2} → ('a', 'b')

# Dictionary casting (requires specific format)

dict\_from\_list = dict([('a', 1), ('b', 2)]) # [('a',1), ('b',2)] → {'a':1, 'b':2}

dict\_from\_tuple = dict((('a', 1), ('b', 2))) # (('a',1), ('b',2)) → {'a':1, 'b':2}

# Set casting

set\_from\_list = set([1, 2, 2, 3]) # [1, 2, 2, 3] → {1, 2, 3}

set\_from\_tuple = set((1, 2, 2, 3)) # (1, 2, 2, 3) → {1, 2, 3}

# Frozenset casting

frozen\_from\_list = frozenset([1, 2, 3]) # [1, 2, 3] → frozenset({1, 2, 3})

frozen\_from\_set = frozenset({4, 5, 6}) # {4, 5, 6} → frozenset({4, 5, 6})

## Operators with Examples

### 1. Arithmetic Operators

# Lists

list1 = [1, 2]

list2 = [3, 4]

print(list1 + list2) # Addition/Concatenation: [1, 2, 3, 4]

print(list1 \* 3) # Multiplication/Repetition: [1, 2, 1, 2, 1, 2]

# Tuples

tuple1 = (1, 2)

tuple2 = (3, 4)

print(tuple1 + tuple2) # (1, 2, 3, 4)

print(tuple1 \* 2) # (1, 2, 1, 2)

# Sets don't support + or \* operations

### 2. Comparison Operators

list1 = [1, 2]

list2 = [1, 2]

list3 = [1, 2, 3]

print(list1 == list2) # True - same values

print(list1 != list3) # True

print(list1 < list3) # True - list1 is shorter

set1 = {1, 2}

set2 = {2, 1}

print(set1 == set2) # True - order doesn't matter in sets

dict1 = {'a': 1, 'b': 2}

dict2 = {'b': 2, 'a': 1}

print(dict1 == dict2) # True - order doesn't matter in dicts

### 3. Python Assignment Operators

# Lists

list1 = [1, 2]

list1 += [3, 4] # Equivalent to list1 = list1 + [3, 4]

print(list1) # [1, 2, 3, 4]

list1 \*= 2 # Equivalent to list1 = list1 \* 2

print(list1) # [1, 2, 3, 4, 1, 2, 3, 4]

# Sets

set1 = {1, 2}

set1 |= {3, 4} # Union assignment (set1.update({3, 4}))

print(set1) # {1, 2, 3, 4}

set1 &= {2, 3} # Intersection assignment (set1.intersection\_update({2, 3}))

print(set1) # {2, 3}

### 4. Logical Operators

list1 = [1, 2]

list2 = []

list3 = [3, 4]

# and - returns first falsy value or last truthy value

print(list1 and list2) # [] (list2 is falsy)

print(list1 and list3) # [3, 4] (both truthy, returns last)

# or - returns first truthy value or last falsy value

print(list1 or list2) # [1, 2] (list1 is truthy)

print(list2 or list1) # [1, 2] (list2 is falsy, returns list1)

# not - returns boolean opposite

print(not list1) # False

print(not list2) # True

### 5. Membership Operators

# in and not in

my\_list = [1, 2, 3, 4]

my\_tuple = (1, 2, 3, 4)

my\_set = {1, 2, 3, 4}

my\_dict = {'a': 1, 'b': 2}

my\_frozenset = frozenset({1, 2, 3})

print(3 in my\_list) # True

print(5 not in my\_tuple) # True

print(3 in my\_set) # True

print('a' in my\_dict) # True (checks keys)

print(1 in my\_dict) # False

print(2 in my\_frozenset) # True

### 6. Identity Operators

list1 = [1, 2, 3]

list2 = [1, 2, 3]

list3 = list1

print(list1 is list2) # False - different objects

print(list1 is list3) # True - same object

print(list1 is not list2) # True

# For immutable types like tuple, Python may reuse objects

tuple1 = (1, 2)

tuple2 = (1, 2)

print(tuple1 is tuple2) # May be True or False (implementation dependent)

### 7. Ternary Operator

my\_list = [1, 2, 3]

result = "Has elements" if my\_list else "Empty"

print(result) # "Has elements"

empty\_set = set()

size = "Non-empty" if len(empty\_set) > 0 else "Empty"

print(size) # "Empty"

### 8. Operator Precedence

From highest to lowest precedence (for data structures):

1. Parentheses () (highest)
2. Subscription x[i], Slicing x[i:j]
3. Attribute reference x.attribute
4. Exponentiation \*\*
5. Bitwise NOT ~x
6. Unary + and - +x, -x
7. Multiplication \*, Division /, Floor division //, Modulus %
8. Addition +, Subtraction -
9. Bitwise shifts <<, >>
10. Bitwise AND &
11. Bitwise XOR ^
12. Bitwise OR |
13. Comparisons in, not in, is, is not, <, <=, >, >=, !=, ==
14. Boolean NOT not
15. Boolean AND and
16. Boolean OR or
17. Conditional expression if-else (lowest)

Example:

result = [1, 2] + [3] \* 2 # Equivalent to [1, 2] + ([3] \* 2) → [1, 2, 3, 3]

print(result)

### 9. Difference between "is" vs "=="

# "==" checks for equality of values

# "is" checks for identity (same object in memory)

list1 = [1, 2, 3]

list2 = [1, 2, 3]

list3 = list1

print(list1 == list2) # True - same values

print(list1 is list2) # False - different objects

print(list1 is list3) # True - same object

# For immutable types, Python may optimize

tuple1 = (1, 2)

tuple2 = (1, 2)

print(tuple1 == tuple2) # True

print(tuple1 is tuple2) # May be True or False (implementation dependent)

# Special case: None, True, False

x = True

print(x is True) # True - always use "is" for singletons like None, True, False

**Key points:**

* Use == when you want to check if two objects have the same value
* Use is when you want to check if two variables reference the exact same object
* For immutable types (like tuples, strings, numbers), == and is may give the same result due to Python's optimization, but don't rely on this
* Always use is when comparing to None, True, or False