Variables & Data Types

Variables

Variables are containers for storing data values.

In Python, you don't need to declare the type.

```
x = 5
name = 'Python'
print(type(x))
print(type(name))
```

Output:

```
>>> <class 'int'>
>>> <class 'str'>
```

Basic Data Types

Numeric Types
- int: 10, -5
- float: 10.5, 3.14
- complex: 1+2j

Sequence Types
- str: 'Hello'
- list: [1, 2, 3]
- tuple: (1, 2, 3)

Operators in Python

Arithmetic Operators

+ (Addition), - (Subtraction), * (Multiplication)

/ (Division), % (Modulus), ** (Exponentiation)

// (Floor Division)

```
a = 10
b = 3
print(a // b) # Floor Division
print(a % b)  # Modulo
```

Output:

```
>>> 3
>>> 1
```

Comparison Operators

== (Equal), != (Not Equal), > (Greater than)

< (Less than), >=, <=

Control Flow - If-Else

Conditional Statements

Python uses indentation to define blocks of code.

```
age = 18
if age >= 18:
    print('You can vote!')
else:
    print('Too young!')
```

Output:

```
>>> You can vote!
```

Common Mistake

Forgetting the colon (:) at the end of if/else statements!

Incorrect indentation.

Loops - Iteration

For Loop

Used for iterating over a sequence.

```
fruits = ['apple', 'banana']
for x in fruits:
    print(x)
```

Output:

```
>>> apple
>>> banana
```

While Loop

Executes as long as a condition is true.

```
i = 1
while i < 3:
    print(i)
    i += 1
```

Output:

```
>>> 1
>>> 2
```

Functions

Defining a Function

A function is a block of code which only runs when it is called.

```
def my_func(name):
    return 'Hello ' + name

print(my_func('Manus'))
```

Output:

```
>>> Hello Manus
```

TIP!
Functions help in code reusability!

Lists & Tuples

Lists (Mutable)

Ordered collection of items. Can be changed.

```
my_list = [1, 2, 3]
my_list.append(4)
print(my_list)
```

Output:

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

Tuples (Immutable)

Ordered collection. CANNOT be changed.

```
my_tuple = (1, 2, 3)
# my_tuple[0] = 10 -> ERROR!
```

Dictionaries

Key-Value Pairs

Dictionaries are used to store data values in key:value pairs.

```
person = {'name': 'Alice', 'age': 25}
print(person['name'])
```

Output:

```
>>> Alice
```

Dictionary Methods

- keys(): Get all keys
- values(): Get all values
- items(): Get key-value pairs

UNIT 8 – SETS

What is a Set?

A set is an unordered collection of items.

Every set element is unique (no duplicates) and

must be immutable.

```
my_set = {1, 2, 3, 3, 2}
print(my_set)
```

Output:

```
>>> {1, 2, 3}
```

Properties of Sets

1. Unordered: No index access
2. Unique: Automatically removes duplicates
3. Mutable: You can add/remove items

Set Methods

Adding & Removing Elements

```
s = {1, 2}
s.add(3)        # Adds 3
s.remove(2)     # Removes 2 (Error if not found)
s.discard(10)   # No error if 10 not found
s.pop()         # Removes random element
```

TIP!
Use discard() to avoid Errors!

Interview Question

Q: Difference between remove() and discard()?

A: remove() raises KeyError if item is missing, discard() does nothing.

Set Operations - I

1. Union (|)

Combines elements from both sets.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A | B)
```

Output:

```
>>> {1, 2, 3, 4, 5}
```

2. Intersection (&)

Common elements in both sets.

```
print(A & B)
```

Output:

```
>>> {3}
```

Set Operations - II

3. Difference (-)

Elements in A but NOT in B.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A - B)
```

Output:

```
>>> {1, 2}
```

4. Symmetric Difference (^)

Elements in either A or B, but NOT in both.

```
print(A ^ B)
```

Output:

```
>>> {1, 2, 4, 5}
```

Sets: Summary & Exercises

Quick Revision
- {} is a dictionary, set() is an empty set!
- Sets are great for removing duplicates.
- Sets are unordered - no indexing!

Mini Exercise

Q1: Given list [1,2,2,3,4,4,5], how to get unique values?

Ans: set([1,2,2,3,4,4,5])

Interview Question

Q: Can a set contain a list?

Ans: No, because lists are mutable (unhashable).

Intermediate: File Handling

Reading & Writing Files

```
# Writing
with open('test.txt', 'w') as f:
    f.write('Hello Python!')

# Reading
with open('test.txt', 'r') as f:
    print(f.read())
```

Output:

```
>>> Hello Python!
```

TIP!
Always use 'with' to auto-close files!

Intermediate: Exception Handling

Try-Except Blocks

Handling errors gracefully.

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print('Cannot divide by zero!')
```

Output:

```
>>> Cannot divide by zero!
```

Intermediate: OOP Basics

Classes & Objects

```
class Student:
    def __init__(self, name):
        self.name = name

s1 = Student('Rahul')
print(s1.name)
```

Output:

```
>>> Rahul
```

Memory Trick

Class = Blueprint (Map of a house)

Object = Actual House built from map

Introduction to Python

What is Python?

Python is a high-level, interpreted, interactive and object-oriented scripting language. It is designed to be highly readable.

TIP!
Created by Guido van Rossum in 1991!

Key Features

1. Easy to Learn & Read
2. Open Source
3. Large Standard Library
4. Platform Independent

Real-world Examples
- Web Dev (Django, Flask)
- Data Science (Pandas, NumPy)
- Automation & Scripting
- AI & Machine Learning