**Python Day 2: Mutable Data Types & Variables**

**Mutable Data Types:** As your notes state, these are data types "whose values can be changed after they're created." You can add, remove, or modify elements within these objects without creating an entirely new object. The types we will cover today—**List, Dictionary, and Set**—are all mutable.

**Core Mutable Data Types**

Let's explore the powerful and flexible collection types in Python.

**1. List (list)**

A List is a versatile, ordered collection of items. Think of it as a grocery list where you can add, remove, and re-order items.

* **Definition:** As your notes describe, a list is a "collection of similar and mix values which is surrounding by Square bracts [] which is separated by coma."
* **Key Properties:**
  + **Mutable:** You can change its content (add, remove, or modify elements).
  + **Ordered:** The items maintain a specific order. You can access them by their position (index).
  + **Allows Duplicates:** A list can contain multiple instances of the same value.

**Code Example: Creating a List**

# List

print([1,2,3,4,5])

print(type([1,2,3,4,5]))

*Output:*

[1, 2, 3, 4, 5]

<class 'list'>

**Explanation**

1. print([1,2,3,4,5]): This line creates a list containing five integers and prints it to the console.
2. print(type([1,2,3,4,5])): The type() function confirms that any collection of items inside square brackets [] is a <class 'list'>.

**Code Example: A List with a Single Value**

# Single value in list

print([1])

print(type([1])

*Output:*

[1]

<class 'list'>

**Explanation**

1. print([1]): This correctly creates and prints a list containing only one element, the integer 1.
2. print(type([1])): This is the **correct way** to find the type. The value [1] is passed inside the function's parentheses (), and it correctly identifies the data type as <class 'list'>.

**2. Dictionary (dict)**

* **Definition:** A dictionary is a "collecting of key's and values pairs witch is surrounding by curly braces {} and separated by coma, keys and values is separated by colane (:)."
* **Key Properties:**
  + **Mutable:** You can add, remove, or change key-value pairs.
  + **Keys are Unique:** A dictionary cannot have duplicate keys. If you assign a new value to an existing key, it will overwrite the old value.
  + **Ordered (in modern Python):** As of Python 3.7+, dictionaries remember the order in which items were inserted.
  + **Unordered (in older Python):** In Python 3.6 and earlier, dictionaries did not maintain any specific order.

**Code Example: Creating Dictionaries and Sets**

It looks like there was a small mix-up in your original notes. Let's clarify it.

# Dictionary

print ({'a':1,'b':2,'c':3,'d':4})

print(type({'a':10,}))

*Output:*

{'a':1,'b':2,'c':3,'d':4}

<class 'dict'>

**Explanation**

1. print({'a','b','c','d'}): When you have comma-separated values inside curly braces {} **without colons**, you create a **Set**, not a dictionary. Notice the output is unordered. We'll cover Sets next!
2. dict({'a':1,'b':2,'c':3,'d':4}): This line correctly creates a dictionary but doesn't do anything with it (like printing or assigning it to a variable), so no output is shown.
3. print(type({'a':10,})): This correctly uses the key:value syntax inside curly braces, creating a dictionary. The type() function confirms it is a <class 'dict'>.

**Code Example: Duplicate Keys in a Dictionary**

This example perfectly demonstrates how dictionaries handle unique keys.

# Dictionry with dublicate values

print({'a':10,'b':20,'a':30})

*Output:*

{'a': 30, 'b': 20}

**Explanation**

1. Python reads the dictionary from left to right:
   * It sees the key 'a' and assigns it the value 10.
   * It sees the key 'b' and assigns it the value 20.
   * It sees the key 'a' again and **overwrites** the previous value with the new value, 30.
2. The final dictionary only contains the last value assigned to each unique key.

**3. Set (set)**

* **Definition:** A set is a "collection of unique elements witch is surrounding by curly braces and separated by Coma."
* **Key Properties:**
  + **Mutable:** You can add or remove elements from a set.
  + **Unordered:** Items in a set do not have a defined order and cannot be accessed by an index.
  + **Unique Elements:** Sets automatically discard any duplicate values.

**Code Example: Creating a Set**

# Set

print({1,2,3,4,5})

print(type({1,2,3,4,5}))

*Output:*

{1, 2, 3, 4, 5}

<class 'set'>

**Code Example: Sets Handle Duplicates**

# Set Dublicate values

print({1,2,3,4,5,1,2,3,4,5})

*Output:*

{1, 2, 3, 4, 5}

**Explanation**

* Even though the initial collection had 10 numbers, the set automatically removed all the duplicates. The final set contains only one instance of each unique element.

**Further Study: Frozenset**

Your homework was to look into the **frozenset**. A frozenset is simply an **immutable** version of a set. Once created, you cannot add or remove elements from it. Because it's immutable, it can be used as a key in a dictionary, whereas a regular set cannot.

**Variables in Depth**

Let's expand on the concept of variables.

**What is a Variable?**

Your notes define a variable perfectly: "it is temporary memory locater where we can store the data."

* **Analogy:** Think of a variable as a labeled box. You can put data inside the box (assignment), and the label (variable name) helps you find that data later.

**Finding a Variable's Memory Location: id()**

The id() function gives you the unique memory address of an object. This address is a unique integer that represents where the data is physically stored in your computer's memory.

**Code Examples: Using id()**

# location int

print(id(110)) # Output: 11645576

# Location with string

print(id('Omkar Dhanke')) # Output: 139343225644848

**Explanation**

* Each call to id() returns the specific memory address for the data (110, 'Omkar Dhanke', and the string 'Omkar' pointed to by a). The exact number will change every time you run the program, but it shows that each piece of data lives at a unique location.

**Variable Naming Rules & Properties**

Your notes highlight two important rules:

1. **Starting Characters:** A variable name must start with a letter (a-z, A-Z) or an underscore (\_). It cannot start with a number.
2. **Case Sensitive:** Python treats uppercase and lowercase letters as different characters. myVar, myvar, and MYVAR are three distinct variables.

**Code Example: Case Sensitivity**

# Case sensitive variable

ABC = 1254

Abc = 464

abc = 784

print(ABC,Abc)

*Output:*

1254 464

**Explanation**

* This code creates three separate variables: ABC, Abc, and abc. Because they have different casings, Python treats them as completely independent "boxes" for storing data. The print statement correctly retrieves the values from the first two variables.

**Variable Swapping Techniques**

Swapping is the process of exchanging the values of two variables. Your notes show two common methods.

**1. Two-Way Swapping (Using Math)**

This method swaps the values of two numeric variables without needing a third variable.

**Code Example**

# 2 way variable swaping

a = 10

b = 20

# Before swap

20

a = a+b # a becomes 10 + 20 = 30

b = a-b # b becomes 30 - 20 = 10

a = a-b # a becomes 30 - 10 = 20

# After swap

# a is 20, b is 10

print(a)

print(b)

*Output:*

20

10

**Explanation**

This is a classic arithmetic trick. By the end of the operations, the original value of b is now in a, and the original value of a is now in b.

**2. Three-Way Swapping (Using a Temporary Variable)**

This is the most common and intuitive way to swap variables. It works for any data type, not just numbers.

**Code Example**

# 3 way Swaping

a = 100

c = 'omkar'

d = a # 1. Store the value of 'a' in a temporary variable 'd' (d is now 100)

a = c # 2. Assign the value of 'c' to 'a' (a is now 'omkar')

c = d # 3. Assign the stored value from 'd' to 'c' (c is now 100)

print(a)

print(c)

*Output:*

omkar

100

**Explanation**

* We use a temporary variable d as a placeholder. It holds the original value of a so it's not lost when we overwrite a with the value of c.