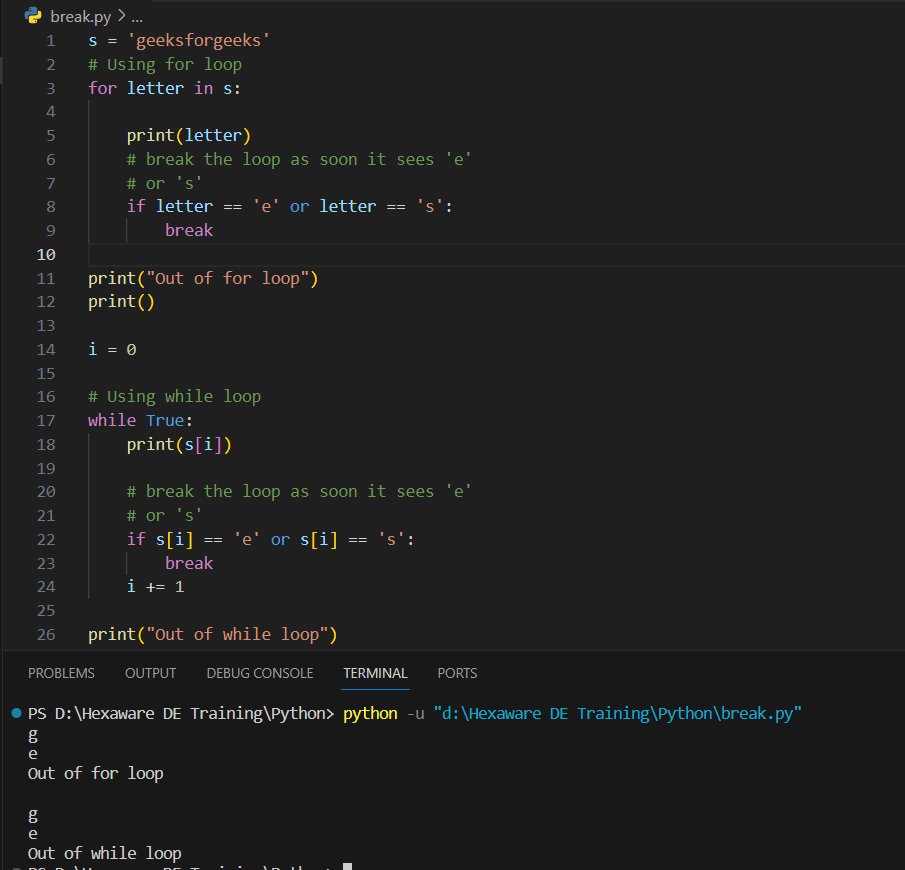
**Day 10 Assignment - 14/12/2023 - Vamsi Viswanadham**

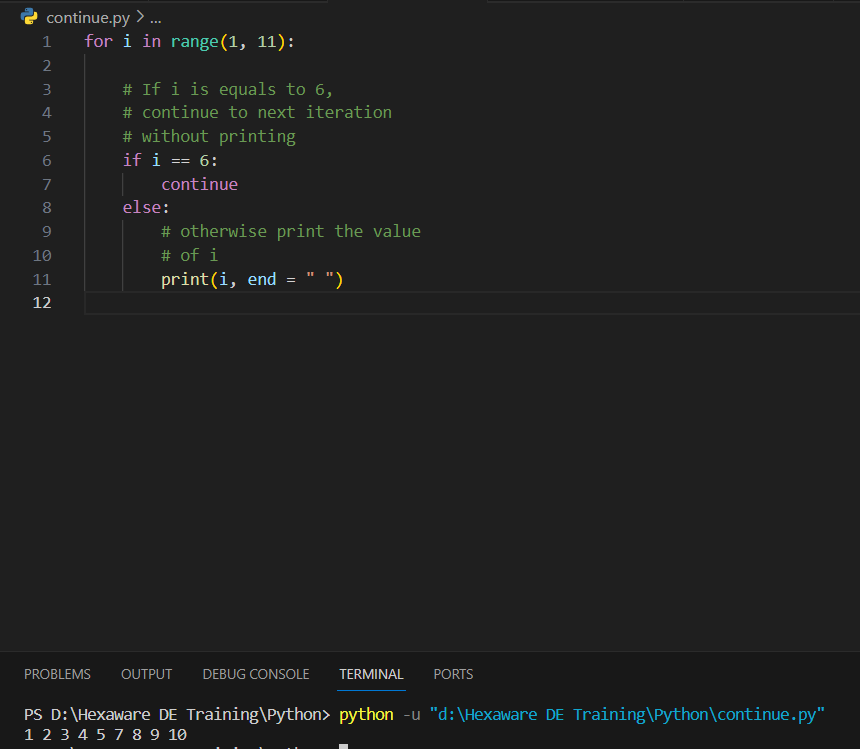
## Break, continue, pass:

**break:** Used to terminate the loop entirely. When break is encountered inside a loop (for or while), control flow breaks out of the loop and proceeds to the first statement following the loop.

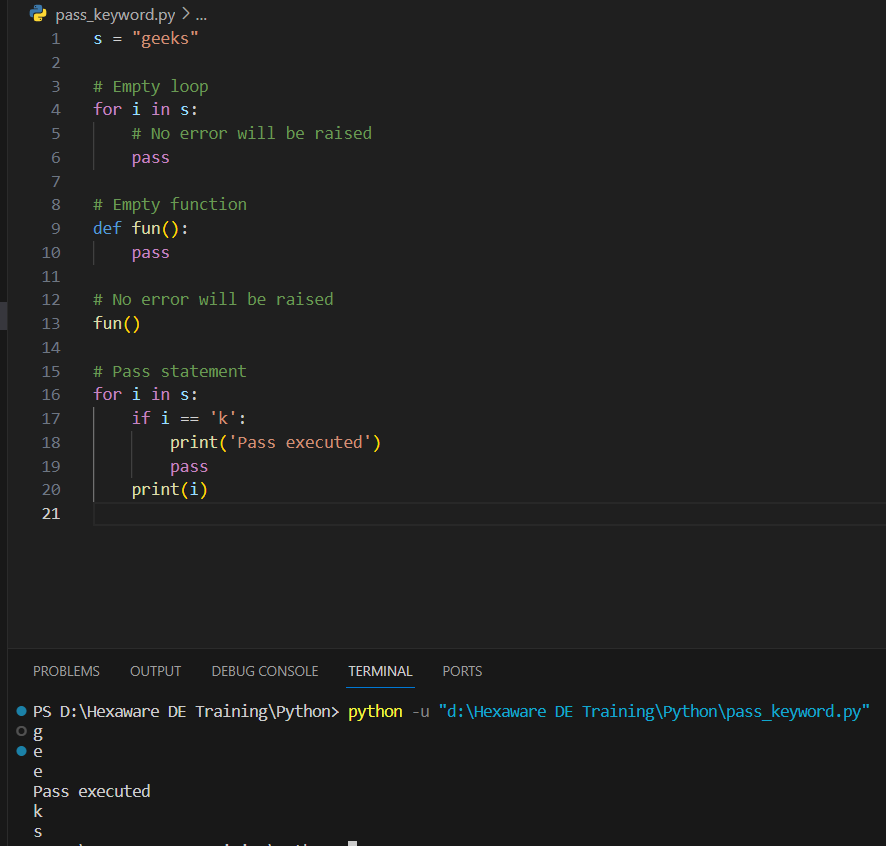
Here is an Example for illustrating break statement



**continue:** Skips the rest of the code inside a loop for the current iteration only. When encountered, the loop does not terminate but immediately proceeds to the next iteration. This is often used to skip an iteration if a certain condition is met, without breaking out of the loop completely.



**pass:** A null statement. It does nothing and is used as a placeholder. In Python, blocks of code cannot be empty, so pass is used in places where a statement is syntactically required, but you do not want any command or code to execute. It is often used as a placeholder for future code.



Bitwise Operators:

Bitwise AND (&): This operator compares each bit of its first operand to the corresponding bit of its second operand. If both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.

Bitwise OR (|): This operator compares each bit of its first operand to the corresponding bit of its second operand. If either bit is 1, the corresponding result bit is set to 1. Otherwise, it's set to 0.

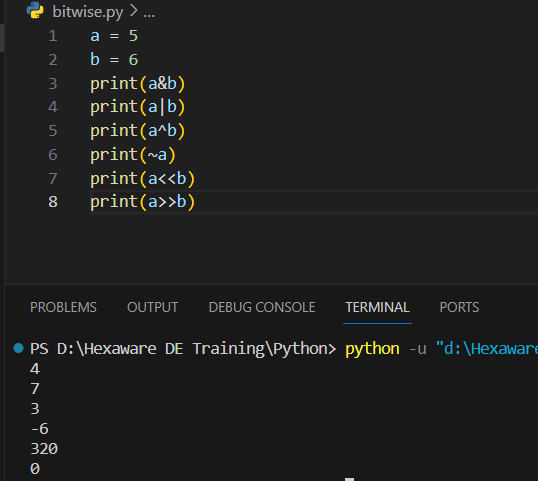
Bitwise XOR (^): This operator is used to compare two bits, setting each bit to 1 if only one of the two bits is 1.

Bitwise NOT (~): This is a unary operator and operates by flipping all the bits in the operand.

Bitwise Left Shift (<<): This operator shifts the bits of the first operand left by the number of positions specified by the second operand. New bits on the right are filled with 0.

Bitwise Right Shift (>>): This operator shifts the bits of the first operand right by the number of positions specified by the second operand. Whether new bits on the left are filled with 0 or 1 depends on the sign of the initial number in many implementations.

Here is an example:



Relational Operators:

Equal to (==): This operator checks whether two values are equal.

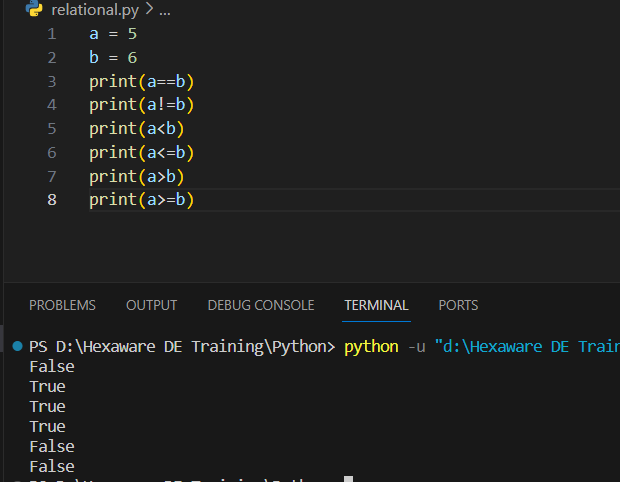
Not equal to (!=): This operator verifies if two values are not equal.

Greater than (>): It determines if the value on the left is greater than the value on the right.

Less than (<): This checks if the value on the left is less than the value on the right.

Greater than or equal to (>=): It checks whether the left value is either greater than or equal to the right value.

Less than or equal to (<=): This operator assesses if the left value is either less than or equal to the right value.



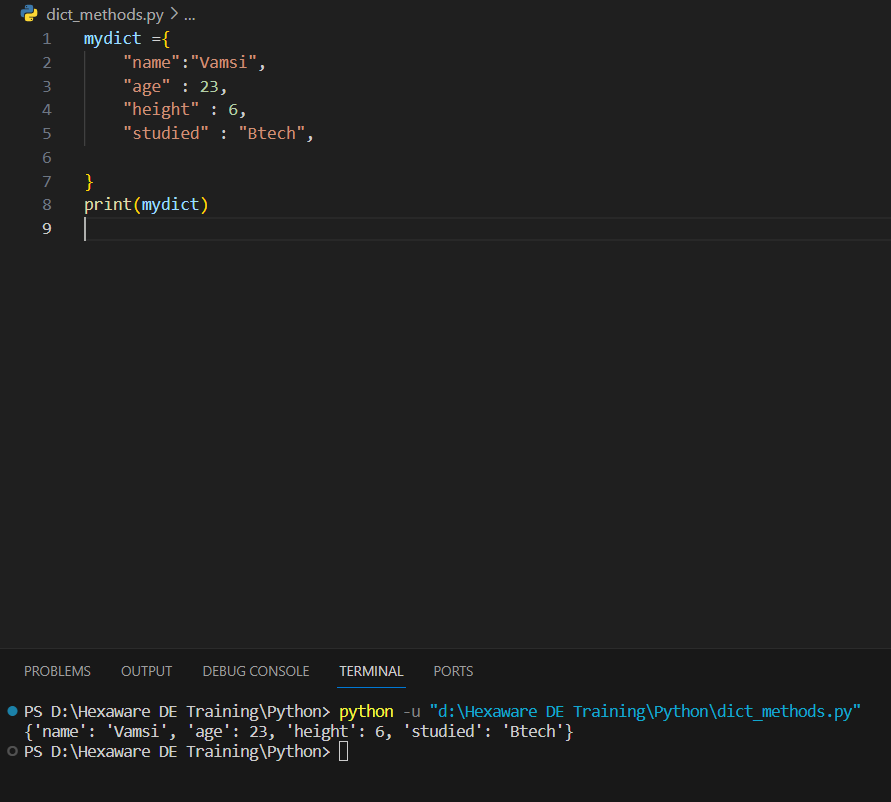
## Dictionary:

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

A dictionary is a collection which is ordered\*, changeable and do not allow duplicates.

Example :

A dictionary named “mydict” is declared here with key-value pairs

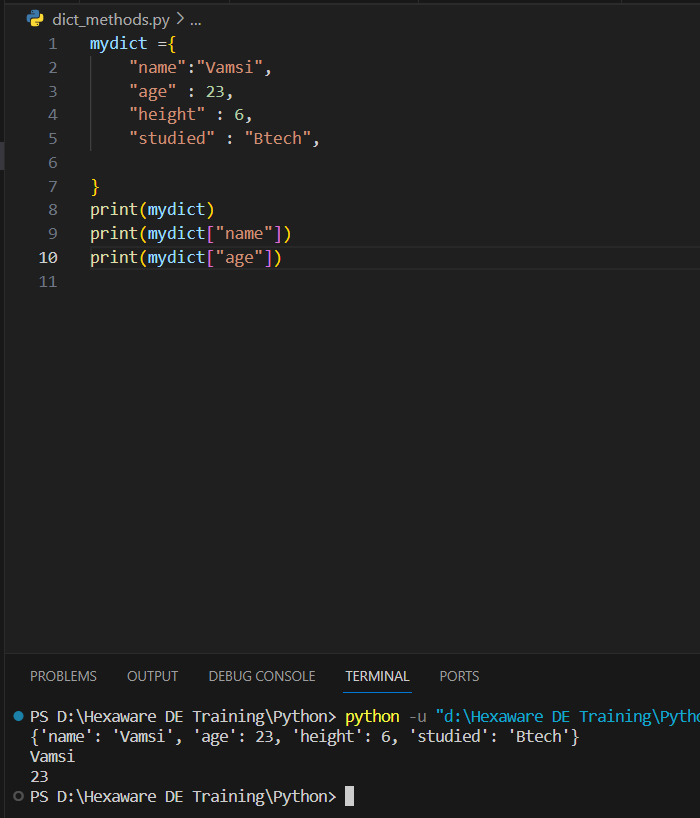


Dictionary Methods :

Dictionary items are ordered, changeable, and does not allow duplicates.

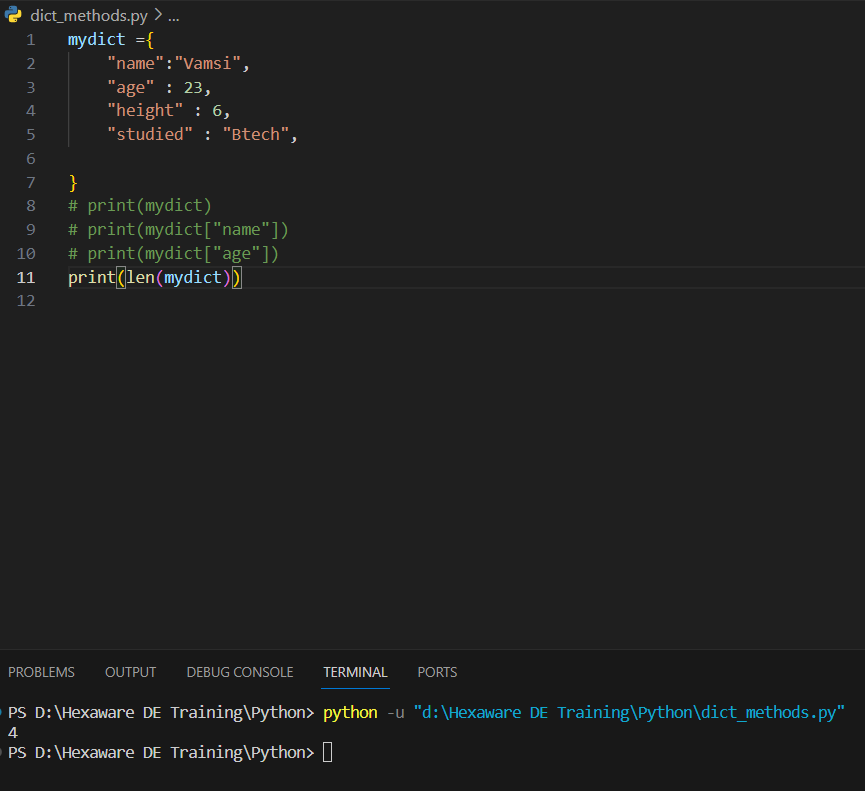
Dictionary items are presented in key:value pairs, and can be referred to by using the key name.

Example:



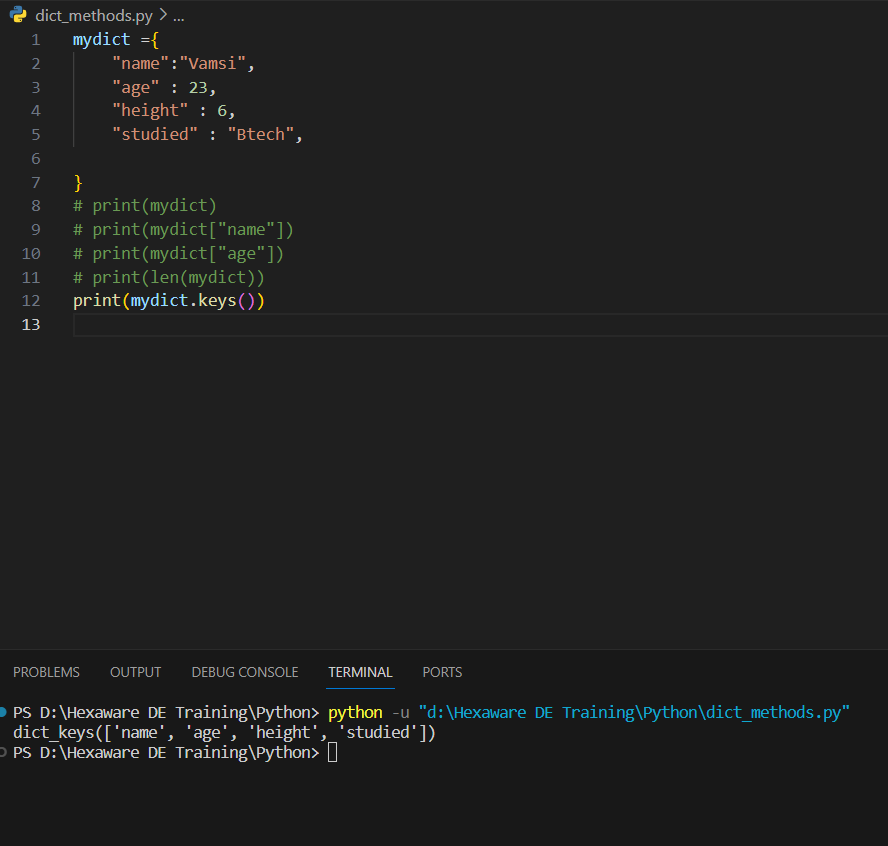
Dictionary Length :

To determine how many items a dictionary has, use the len() function:



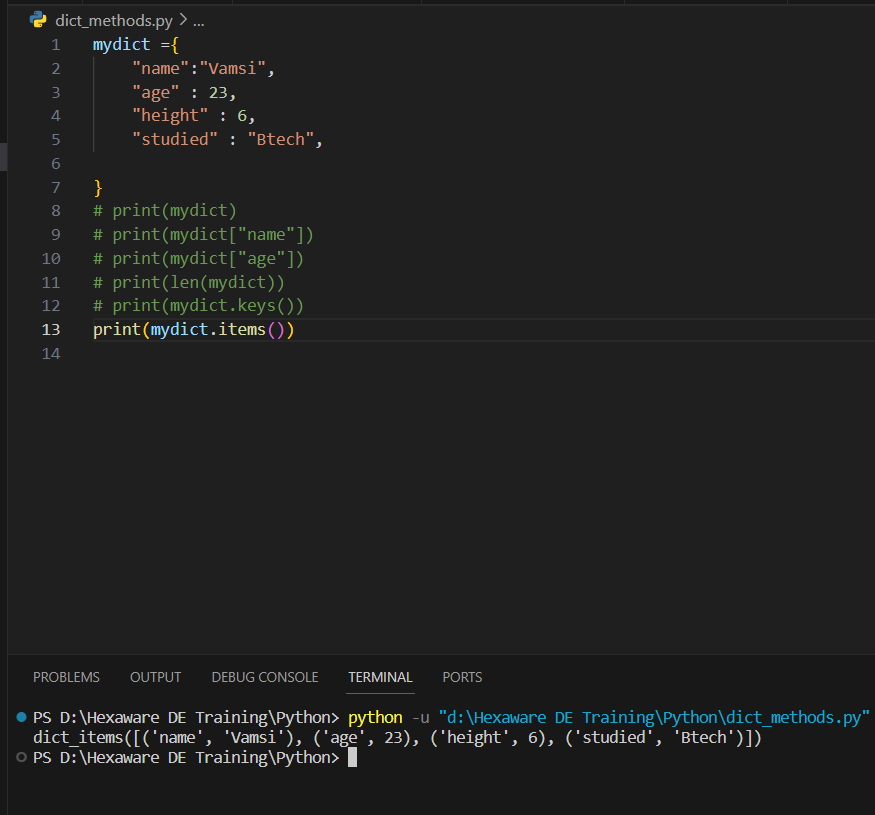
Get Keys Method:

The keys() method will return a list of all the keys in the dictionary.



Get Items:

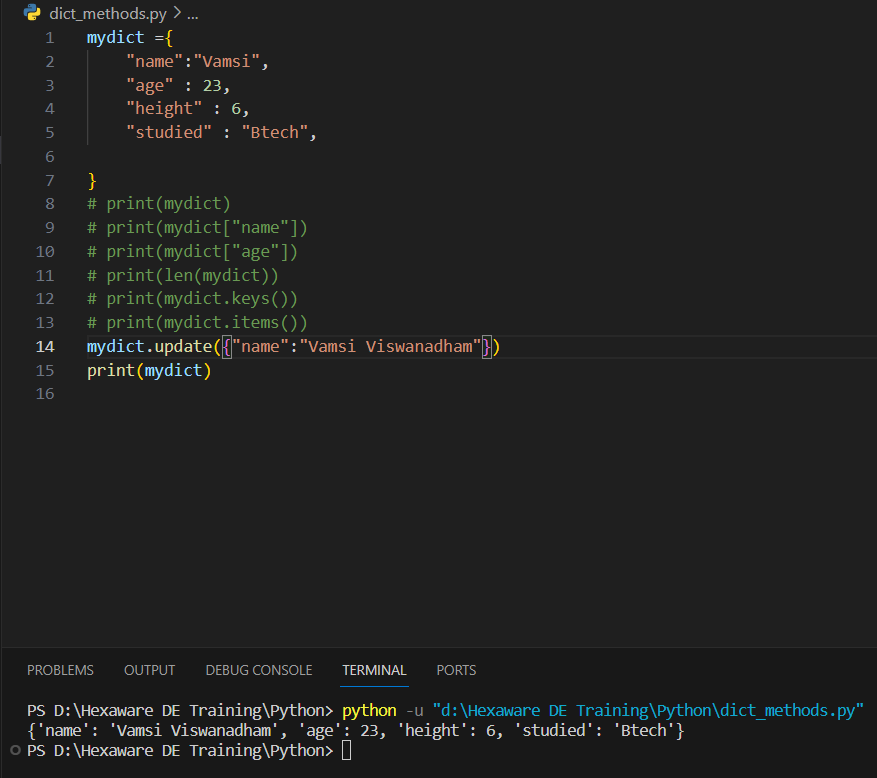
The items() method will return each item in a dictionary, as tuples in a list.



Update Dictionary Method :

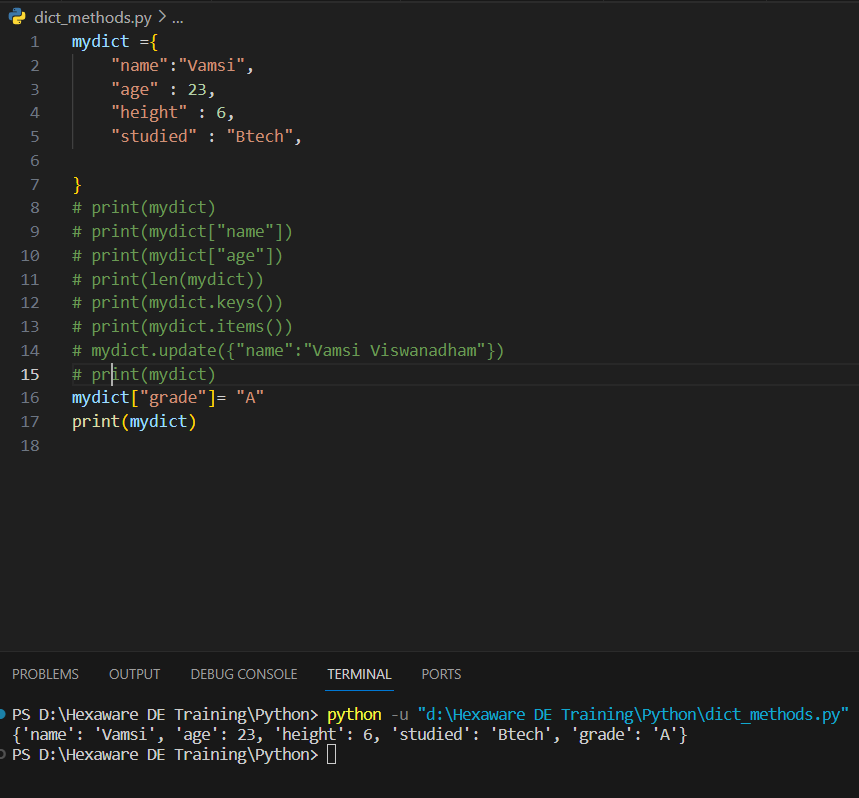
The update() method will update the dictionary with the items from the given argument.

The argument must be a dictionary, or an iterable object with key:value pairs.



Adding Items :

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

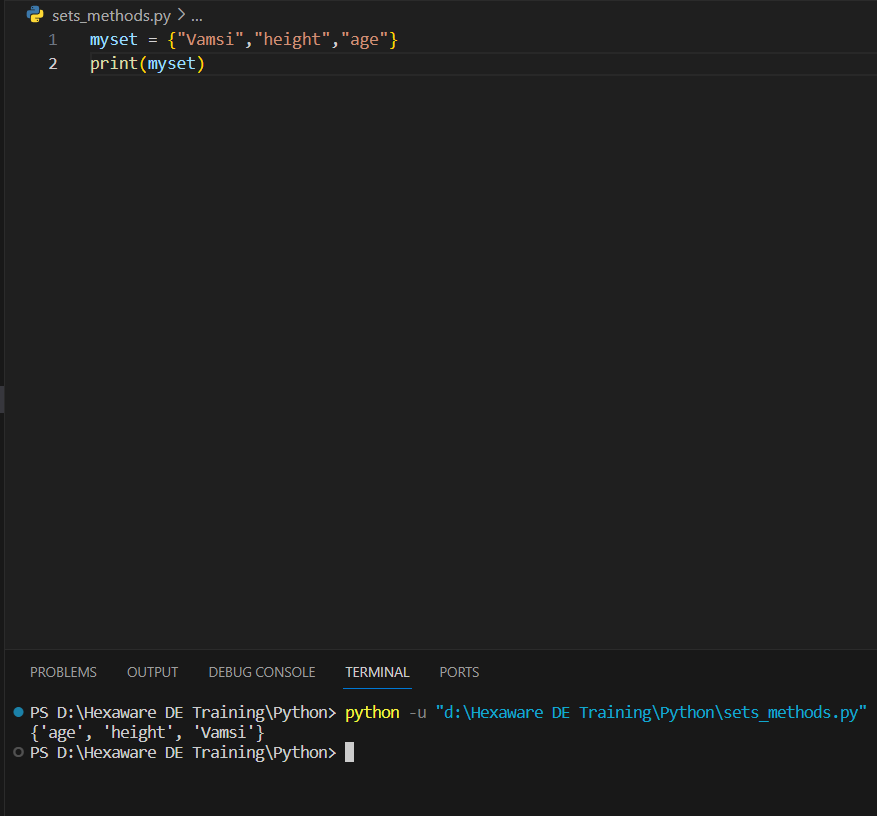


Set :

Sets are used to store multiple items in a single variable.

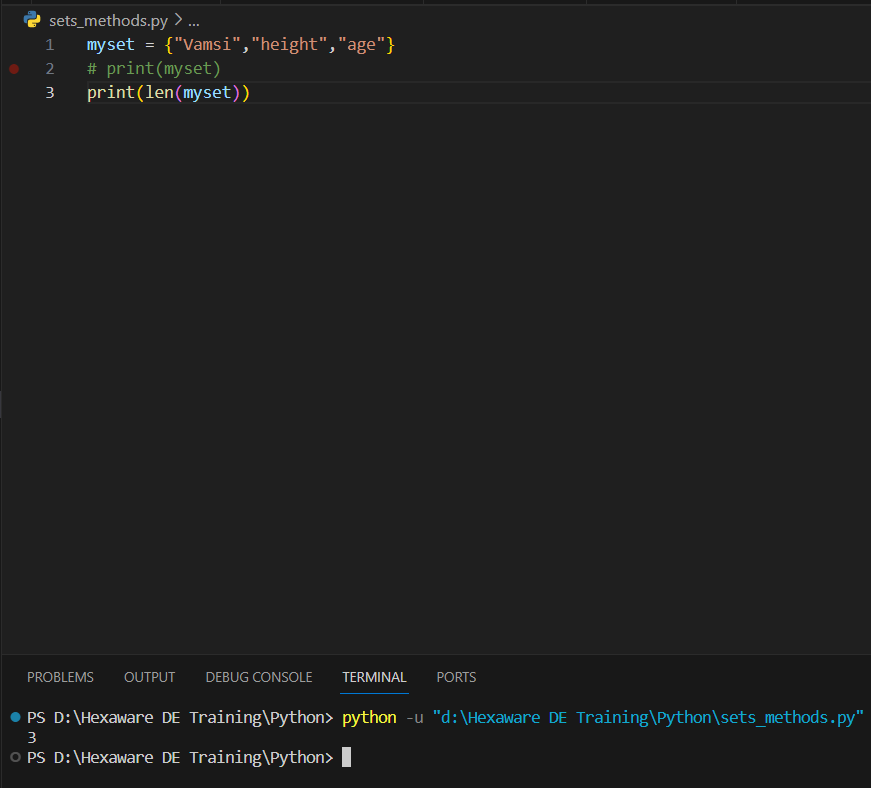
A set is a collection which is unordered, unchangeable\*, and unindexed.

Example:



Length of a Set :

To determine how many items a set has, use the len() function.



Access Items:

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in keyword.