# More Exercises: Data Types and Variables

Additional exercises for the [Python Fundamentals Course @SoftUni](https://softuni.bg/trainings/3368/python-fundamentals-may-2021).

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1723>.

***Note: All the exercises are excluded from your homework!***

1. **Exchange Integers**

Read two integer numbers and after that **exchange their values**. Print the variable values before and after the exchange, as shown below:

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  10 | Before:  a = 5  b = 10  After:  a = 10  b = 5 |

**Hints**

You may use a **temporary variable** to remember the old value of **a**, then assign the value of **b** to **a**, then assign the value of the temporary variable to **b**.

a = old\_a = input()  
b = old\_b = input()  
  
print(f"Before:")  
print(f"a = {a}")  
print(f"b = {b}")  
a = old\_b  
b = old\_a  
print(f"After:")  
print(f"a = {a}")  
print(f"b = {b}")

## Prime Number Checker

Write a program to check if a number is **prime**. A prime number is a natural number greater than 1 that is not a product of two smaller natural numbers. For example, the only ways of writing 5 as a product, 1 × 5 or 5 × 1, involve 5 itself.The **input** comes as an integer number.

The **output** should be **True** if the number is prime and **False** otherwise.

num = int(input())  
prime = True  
  
if num > 1:  
 for i in range(2, num):  
 if (num % i) == 0:  
 prime = False  
 break

print(prime)

**Examples**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |  | **Input** | **Output** |
| 7 | True |  | 8 | False | 81 | False |

## Decrypting Messages

On the **first line**, you will receive a **key** (**integer**). On the **second line**, you will receive **n** – the number of **lines**, which will **follow**.On the next **n lines** – you will receive a **lower** and an **uppercase** letter per line.

You should **add the key** to **each of the characters** and append them to a **message**. At the end **print the** **decrypted message**.

key\_integer = int(input())  
number\_of\_lines = int(input())  
message = ""  
  
for i in range(0, number\_of\_lines):  
 encrypted\_letter = input()  
 decrypted\_letter = ord(encrypted\_letter) + key\_integer  
 message += chr(decrypted\_letter)  
  
print(message)

### Examples

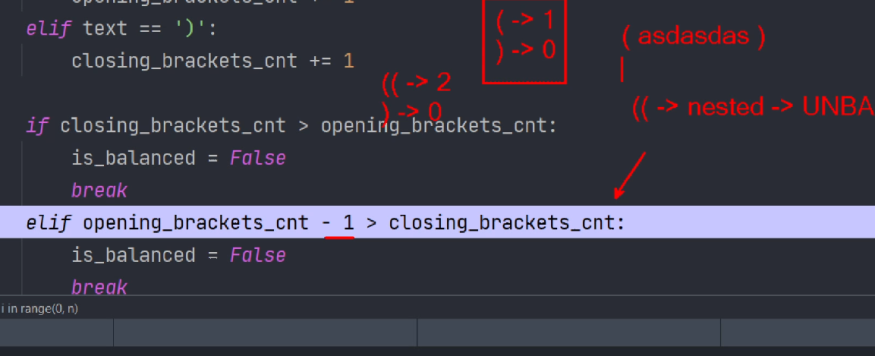
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| **3**  **7**  P  l  c  q  R  k  f | SoftUni |  | **1**  **7**  C  d  b  q  x  o  s | Decrypt |

## Balanced Brackets

On the **first line**, you will receive **n** – the number of lines, which will follow. On the next **n** lines, you will receive **one** of the following:

* Opening bracket – “(“,
* Closing bracket – “)” or
* **Random string**

Your task is to find out if the **brackets** are **balanced**. That means after every **closing** bracket should follow an **opening** one. Nested parentheses are **not valid**, and if **two** **consecutive opening brackets** exist, the expression should be marked as **unbalanced**. You should print “BALANCED”, if the parentheses are balanced and “UNBALANCED” otherwise.

****

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| **8**  (  5 + 10  (  \* 2 +  )  5  )  -12 | BALANCED |  | **6**  12 \*  )  10 + 2 -  (  5 + 10  ) | UNBALANCED |

number\_of\_lines = int(input())  
opening\_brackets\_count = 0  
closing\_brackets\_count = 0  
is\_balanced = True  
  
for i in range(1, number\_of\_lines + 1):  
 line\_string = input()  
 if line\_string == "(":  
 opening\_brackets\_count += 1  
 elif line\_string == ")":  
 closing\_brackets\_count += 1  
 if closing\_brackets\_count > opening\_brackets\_count:  
 is\_balanced = False  
 break  
 elif opening\_brackets\_count - 1 > closing\_brackets\_count:  
 is\_balanced = False  
 break  
  
if opening\_brackets\_count == closing\_brackets\_count:  
 is\_balanced = True  
else:  
 is\_balanced = False  
  
if is\_balanced:  
 print("BALANCED")  
else:  
 print("UNBALANCED")

number\_of\_lines = int(input())  
opening\_brackets\_count = 0  
closing\_brackets\_count = 0  
  
for i in range(1, number\_of\_lines + 1):  
 line = input()  
 open\_bracket= "("  
 closing\_bracket = ")"  
 if line.\_\_contains\_\_(open\_bracket):  
 opening\_brackets\_count += 1  
 if line.\_\_contains\_\_(closing\_bracket):  
 closing\_brackets\_count += 1  
 if opening\_brackets\_count - closing\_brackets\_count != 0:  
 break  
  
if opening\_brackets\_count == closing\_brackets\_count:  
 print("BALANCED")  
else:  
 print("UNBALANCED")