In Python and many other programming languages, objects can be classified as *mutable* or *immutable*. *Mutable* objects are capable of changing their state or value after their creation, while *immutable* objects cannot be changed after they are created.

For instance, mutable objects like lists can be modified inside a function, and this change is reflected in the object outside the function as well. However, for immutable objects like strings, any modification attempt inside a function results in an error, leaving the original object unaffected.

Consider the following examples to better understand this:

```
def mutable_list_1(n, l=[]): # initializing the list
    l.append(n)
    return l

def mutable_list_2(n):
    l = None
    l = []
    l.append(n)
    return l

def immutable_string(st):
    try:
        st[0] = 'e'
        return st
    except Exception as e:
        return e

print(mutable_list_1(5))    # Outputs: [5]
print(mutable_list_1(10))   # Outputs: [5, 10]

print(mutable_list_2(5))    # Outputs: [5]
print(mutable_list_2(10))   # Outputs: [10]

st = 'old'
print(immutable_string(st))  # Outputs: 'str' object does not support item as
print(st)  # Outputs: 'old'
```

In the function `mutable_list_1`, a list is initialized as a default argument. Lists are mutable objects in Python, so when the function is called multiple times, the changes persist across calls. That's why the function outputs a growing list across successive calls.

The function `mutable_list_2` demonstrates that a new list is created in each function call. So, the list from the previous function call does not affect the next function call.

The function `immutable_string` tries to change the first character of a string, which is an immutable object in Python. Consequently, it results in an error, showcasing the immutability of strings.