تاپل‌ها (Tuples) در پایتون

تاپل‌ها (Tuples) یکی از **ساختارهای داده‌ای** در پایتون هستند که شباهت زیادی به لیست‌ها دارند، اما **غیرقابل تغییر (Immutable)** هستند. این ویژگی باعث می‌شود که تاپل‌ها برای ذخیره داده‌هایی که نباید تغییر کنند، انتخاب مناسبی باشند.

---

۱. تعریف تاپل و تفاوت با لیست

✅ ایجاد تاپل

تاپل‌ها مانند لیست‌ها با استفاده از **ویرگول (`,`)** جدا می‌شوند، اما به جای `[]` (براکت مربع)، از `()` **(پرانتز گرد)** استفاده می‌شود:

```
# تعریف یک تاپل
numbers = (1, 2, 3, 4, 5)

# تاپل شامل انواع داده‌ای مختلف
person = ("Ali", 25, True)

# تاپل خالی
empty_tuple = ()

# تاپل با یک مقدار (باید یک ویرگول داشته باشد!)
single_element_tuple = (42,)
print(type(single_element_tuple))  # خروجی: <class 'tuple'>
```

📌 **نکته:**
اگر `()` بدون ویرگول استفاده شود، به‌عنوان یک مقدار عادی در نظر گرفته می‌شود و تاپل محسوب نمی‌شود.

```
not_a_tuple = (42)  # مقدار معمولی (int)
print(type(not_a_tuple))  # خروجی: <class 'int'>
```

---

۲. تفاوت‌های تاپل و لیست

ویژگی	لیست (List)	تاپل (Tuple)
قابل تغییر (Mutable)	✅ بله (می‌توان مقدار را تغییر داد)	❌ خیر (پس از ایجاد، مقدارها تغییر نمی‌کنند)
سرعت	کندتر	سریع‌تر
حجم حافظه	بیشتر	کمتر
امکان استفاده به عنوان کلید دیکشنری	❌ خیر	✅ بله
کاربرد	داده‌های پویا و قابل تغییر	داده‌های ثابت و بدون تغییر

---

۳. دسترسی به عناصر تاپل

مانند لیست‌ها، می‌توان با استفاده از **ایندکس** به عناصر تاپل دسترسی داشت:

```
colors = ("قرمز", "آبی", "سبز")

print(colors[0])  # خروجی: قرمز
print(colors[1])  # خروجی: آبی
print(colors[-1]) # خروجی: سبز (ایندکس منفی)
```

📌 **نکته:** چون تاپل **غیرقابل تغییر** است، نمی‌توان مقدار عناصر آن را تغییر داد:

```
numbers = (10, 20, 30)
numbers[1] = 50  # ❌ خطا: TypeError: 'tuple' object does not support item assignment
```

---

۴. برش (Slicing) تاپل

مانند لیست‌ها، می‌توان از **برش (slicing)** برای دریافت بخش خاصی از تاپل استفاده کرد:

```
numbers = (10, 20, 30, 40, 50)

print(numbers[1:4])  # خروجی: (20, 30, 40)
print(numbers[:3])   # خروجی: (10, 20, 30)
print(numbers[2:])   # خروجی: (30, 40, 50)
print(numbers[::-1]) # خروجی: (50, 40, 30, 20, 10) ← معکوس کردن تاپل
```

---

۵. عملیات روی تاپل‌ها

✅ `len()` ← تعداد عناصر تاپل

```
numbers = (1, 2, 3, 4, 5)
print(len(numbers))  # خروجی: 5
```

✅ `count(value)` ← تعداد دفعات تکرار یک مقدار

```
numbers = (1, 2, 3, 2, 2, 4)
print(numbers.count(2))  # خروجی: 3
```

✅ `index(value)` ← پیدا کردن ایندکس یک مقدار

```
numbers = (10, 20, 30, 40)
print(numbers.index(30))  # خروجی: 2
```

---

۶. تبدیل لیست به تاپل و برعکس

گاهی اوقات نیاز است که **لیست را به تاپل** یا **تاپل را به لیست** تبدیل کنیم.

✅ تبدیل لیست به تاپل

```
my_list = [1, 2, 3]
my_tuple = tuple(my_list)
print(my_tuple)  # خروجی: (1, 2, 3)
```

✅ تبدیل تاپل به لیست

```
my_tuple = (1, 2, 3)
my_list = list(my_tuple)
print(my_list)  # خروجی: [1, 2, 3]
```

---

۷. تاپل‌های تودرتو (Nested Tuples)

یک تاپل می‌تواند شامل تاپل‌های دیگر باشد.

```
nested_tuple = ((1, 2, 3), ("a", "b", "c"))

print(nested_tuple[0])    # خروجی: (1, 2, 3)
print(nested_tuple[1][2]) # خروجی: c
```

---

۸. استفاده از تاپل در بازگردانی چند مقدار از تابع

توابع در پایتون می‌توانند چند مقدار را با استفاده از **تاپل** برگردانند.

```
def get_person():
    name = "Ali"
    age = 25
    return name, age  # این مقادیر به صورت تاپل برمی‌گردند

person = get_person()
print(person)  # خروجی: ('Ali', 25)

# دسترسی به مقدارها
name, age = get_person()
print(name)  # خروجی: Ali
print(age)   # خروجی: 25
```

---

۹. تاپل‌های تک‌عنصری (Single-Element Tuples)

یک تاپل با یک مقدار باید **یک ویرگول داشته باشد، وگرنه تاپل محسوب نمی‌شود**:

```
wrong_tuple = (42)  # این یک مقدار عددی است، نه تاپل!
print(type(wrong_tuple))  # خروجی: <class 'int'>

correct_tuple = (42,)
print(type(correct_tuple))  # خروجی: <class 'tuple'>
```

---

۱۰. چرا از تاپل استفاده کنیم؟

✅ **امنیت داده‌ها:** از آنجایی که مقدارهای تاپل تغییر نمی‌کنند، **ایمن‌تر** هستند.
✅ **سرعت بیشتر:** پردازش تاپل‌ها **سریع‌تر** از لیست‌ها است.
✅ **حافظه کمتر:** تاپل‌ها **کمتر از لیست‌ها حافظه مصرف می‌کنند.**
✅ **استفاده به عنوان کلید دیکشنری:** از تاپل‌ها می‌توان در **کلیدهای دیکشنری** استفاده کرد، اما لیست‌ها را نمی‌توان استفاده کرد.

```
my_dict = {
    (1, 2): "point",
    (3, 4): "location"
}
print(my_dict[(1, 2)])  # خروجی: point
```

نتیجه‌گیری

- **تاپل‌ها (Tuples)** مانند لیست‌ها هستند اما **غیرقابل تغییر (Immutable)** هستند.
- برای **ذخیره داده‌های ثابت و تغییرناپذیر** استفاده می‌شوند.
- **سرعت و بهینه‌بودن حافظه** از مزایای آن‌ها است.
- قابلیت استفاده در **کلیدهای دیکشنری** را دارند.
- می‌توان از آن‌ها برای **بازگردانی چند مقدار از توابع** استفاده کرد.

📌 **در مواقعی که نیاز به تغییر داده‌ها نیست، تاپل‌ها انتخاب بهتری نسبت به لیست‌ها هستند!** 🚀