# آموزش نصب پایتون در ویندوز

**برای نصب پایتون روی ویندوز، مراحل زیر را دنبال کنید :**

**مرحله :۱ دانلود پایتون**

- به وبسایت رسمی پایتون بروید www.python.org.
- از منوی اصلی، گزینه Downloads را انتخاب کنید
- .نسخه ویندوز مناسب سیستم خود (۳۲ بیتی یا ۶۴ بیتی) را دانلود کنید. معمولاً برای سیستمهای جدیدتر، نسخه ۶۴ بیتی مناسب است .

**مرحله :۲ اجرای فایل نصبی**

- فایل نصبی دانلودشده (معمولاً با نامی شبیهexe.x.x-3.python) را باز کنید
- در پنجره نصب، گزینه Add python to PATH را حتماً تیک بزنید (این کار بسیار مهم است و دسترسی به پایتون از خط فرمان را آسان میکند).
- روی دکمه Customize Installation یا Install Nowرا کلیک کنید
- Install Now : پایتون با تنظیمات پیشفرض نصب میشود
- Customize Installation :. امکان انتخاب ویژگیها و مسیر نصب به شما داده میشود.

**مرحله :۳ اتمام نصب**

- پس از اتمام فرآیند نصب، پنجرهای نمایش داده میشود که گزینه Disable path length limit را ارائه میدهد. روی آن کلیک کنید (اختیاری، اما پیشنهاد میشود).
- روی دکمهClose کلیک کنید

**مرحله :۴ بررسی نصب**

- :Command Prompt (CMD) اجرای

  کلیدهایWindows + R را فشار دهید، سپسcmd را تایپ کرده و Enter بزنید

- دستور زیر را اجرا کنید :

```
python --version
```

اگر نسخه پایتون مثل x.x3. Python) نمایش داده شد، نصب به درستی انجام شده است.

# آموزش نصبVSCode

- **دانلود VSCode :** به وب سایت رسمی VSCode بروید ، نسخه ویندور مناسب سیستم خود (۶۴ بیتی یا ۳۲ بیتی) را دانلود کنید.

- **اجرای فایل نصبی :** فایل دانلود شده (VSCode Setup-x.x.x.exe)را اجرا کنید . با شرایط و ضوابط موافقت کنید (تیک گزینه I accept the agreement ).

- **انتخاب مسیر نصب :** مسیر پیش فرض یا مسیر دلخواه خود را انتخاب کنید.

- **تنظیمات اضافی :** گزینه های زیر را تیک بزنید:

**Add to patch (recommended)**
**Register Code as an editor for supported file types**
**Add a shortcut to the desktop**

**نصب نرم افزار :** روی Install کلیک کنید و منتظر بمانید تا فرایند نصب کامل شود.

- **اجرای VSCode** : گزینه Launch Visual Studio Code را تیک بزنید و روی Finish کلیک کنید.

- **نصب افزونه پایتون :** از منوی Command palette (کلید ctrl + shift + p ) گزینه python: select Interpreter را انتخاب کنید . مسیر مفسر پایتون نصب شده روی سیستم خود را انتخاب کنید.

# آشنایی با ساختار پایه پایتون در کدنویسی

### خطوط کد (Statements) :

- کدها به صورت خطی نوشته میشوند و هر خط یک دستور یا بیان (Statement) است.
- نیازی به علامت نقطه ویرگول در انتهای خطوط نیست (مگر اینکه چند دستور در یک خط نوشته شود).

### تورفتگی(Indentation) :

- پایتون از تورفتگی برای تعریف بلاک های کد استفاده میکند (به جای براکت {} در زبان هایی مثل c یا java)
- تمام خطوط مربوط به یک بلاک باید دارای تورفتگی یکسان باشند .
- استفاده نادرست از تورفتگی منجر به خطا میشود.

### کامنت ها (Comments) :

- برای مستندسازی و توضیح کد استفاده میشوند و توسط مفسر اجرا نمیشوند .
- کامنت تک خطی : با # شروع میشود
- کامنت چند خطی : از سه تا نقل قول(' ' ') یا (" " " ) استفاده میشود .

### متغییرها و انواع داده ها :

- نیازی به تعریف نوع متغییر نیست ؛ پایتون به صورتت پویا نوع داده را تشخیص میدهد .
- از نام های معنادار برای متغییرها استفاده کنید .

### ورودی و خروجی (Input / Output) :

- از تابع input برای دریافت داده از کاربر و از print برای نمایش خروجی استفاده میشود .

### کنترل جریان برنامه :

شامل شرط های ( if، else if) و حلقه های (while , for )

### توابع (Functions ) :

از توابع برای دسته بندی کدهای تکراری استفاده میشود .

### کتابخانه ها و ماژول ها :

میتوانید از ماژول های داخلی یا کتابخانه های خارجی برای گسترش قابلیت ها استفاده کنید .پ

## مفهوم مفسر( Interpreter) و کامپایلر(Compiler) :

در دنیای برنامه نویسی، مفسر و کامپایلر ابزارهایی هستند که کدهای نوشتهشده توسط برنامه نویس را به زبانی تبدیل میکنند که کامپیوتر میتواند آن را بفهمد و اجرا کند. این دو ابزار عملکرد مشابهی دارند، اما روش کارشان متفاوت است. اجازه بدهید به صورت ساده و گام به گام توضیح بدهم :

## مفسر (Interpreter) :

- مفسر یک ابزار نرم افزاری است که کدهای نوشته شده در یک زبان برنامه نویسی را خط به خط میخواند ، تجربه میکند و بلافاصله اجرا میکند. در واقع مفسر کد را خط به خط اجرا میکند و اگر در حین اجرای برنامه خطایی رخ دهد اجرای کد متوقف میشود و خطا را گزارش میدهد .
- زبان های پایتون ، PHP جاوا اسکریپت ، روبی ، پرل و شل اشکریپت مفسری هستند.

## ویژگی های مفسر :

- **اجرای خط به خط :** کد را خط به خط میخواند و اجرا میکند .
- **خطایابی ساده تر :** به محض بروز خطا ، اجرای کد متوقف میشود و خطا را نمایش میدهد .
- **کندی اجرا :** از آنجا که ک خط به خط اجرا میشود ، اجرای برنامه ها ممکن است کمی کندتر باشد.

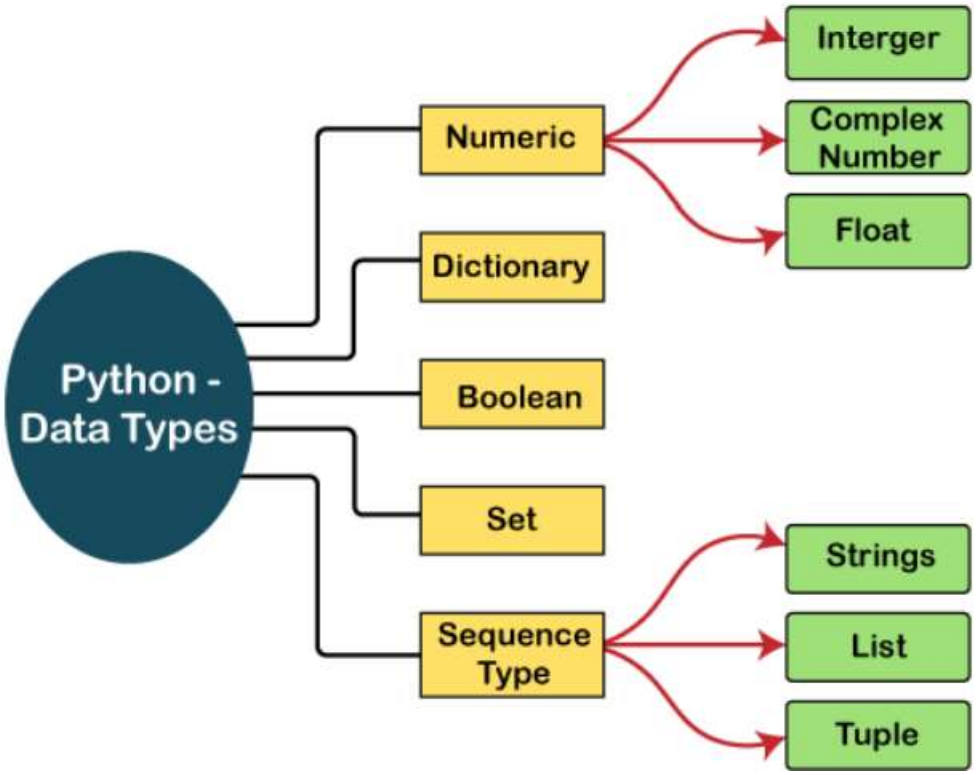
## کامپایلر (Compiler) :

- کامپایلر نیز مانند مفسر ، کدهای برنامه نویس را به کدی تبدیل میکند که کامپیوتر میتواند آن را اجرا کند ، اما کامپایلر تمام کد را در یک مرحله تجزیه و تحلیل کرده و سپس تمام آن را به یک فایل اجرایی تبدیل میکند. این به این معنی است که قبل از اجرا ، کامپایلر کل کد را بررسی میکند و در صورتی که خطایی وجود داشته باشد ، تمام خطاها را گزارش میدهد و کد اجرا نمیشود تا زمانی که خطاها اصلاح شوند.
- زبان های C و C++ ، جاوا ، سی شارپ (C#)، GO ، Swift ، Rust کامپایلری هستند.

## ویژگی های کامپایلر :

- **تجزیه کد به صورت یک پارچه :** تمام کد را به یک فایل اجرایی تبدیل میکند
- **اجرای سریع تر :** پس از کامپایل کردن کد ، میتوانید آن را به سرعت اجرا کنید .
- **خطایابی دشوارتر :** ممکن است چندین خطا در کد وجود داشته باشد که باید پس از کامپایل گزارش شوند.

## انواع داده در پایتون :




## عملگرها در پایتون (operators) :

در پایتون ، عملگرها ابزارهایی هستند که برای انجام عملیات های مختلف روی داده ها (مثل اعداد ، رشته ها ، و غیره) استفاده میشوند . این عملگرها به دسته های مختلفی تقسیم میشوند که در اینجا آنها را به تفصیل توضیح خواهیم داد .

### ۱- عملگر های ریاضی (Arithmetic Operators) :

عملگرهای ریاضی برای انجام عملیات پایه ای مانند جمع ، تفریق ، ضرب و تقسیم استفاده میشوند .

| عملگر | توضیح | مثال | خروجی |
|---|---|---|---|
| + | جمع | 5 + 3 | 8 |
| - | تفریق | 5 - 3 | 2 |
| * | ضرب | 5 * 3 | 15 |
| / | تقسیم (اعشاری) | 5 / 2 | 2.5 |
| // | تقسیم صحیح (بدون اعشار) | 5 // 2 | 2 |
| % | باقی‌مانده تقسیم (مدول) | 5 % 2 | 1 |
| ** | توان | 5 ** 2 | 25 |

## ۲- عملگرهای مقایسه ای (Comparison Operators) :

عملگرهای مقایسه ای برای مقایسه دو مقدار استفاده میشوند و نتیجه آنها یک مقدار بولی ( False و یا True) است .

| عملگر | توضیح | مثال | خروجی |
|---|---|---|---|
| == | برابر با | 5 == 3 | False |
| != | نابرابر با | 5 != 3 | True |
| > | بزرگ‌تر از | 5 > 3 | True |
| < | کوچکتر از | 5 < 3 | False |
| >= | بزرگ‌تر یا مساوی با | 5 >= 3 | True |
| <= | کوچکتر یا مساوی با | 5 <= 3 | False |

## ۳- عملگرهای منطقی (Logical Operators) :

عملگرهای منطقی برای انجام عملیات های منطقی مانند AND و OR و NOT بر روی مقادیر بولی استفاده میشوند .

| عملگر | توضیح | مثال | خروجی |
|---|---|---|---|
| and | اگر هر دو شرط Trueباشند، نتیجه Trueاست | True and False | False |
| or | اگر یکی از شرایط Trueباشد، نتیجه Trueاست | True or False | True |
| not | معکوس کردن مقدار بولی | not True | False |

## ۴- عملگرهای انتساب (Assignment Operators) :

عملگرهای انتساب برای انتساب مقدار به متغییرها استفاده میشوند . این عملگرها میتوانند به شکلی کوتاه تر انجام شوند.

| عملگر | توضیح | مثال | خروجی |
|---|---|---|---|
| = | انتساب مقدار به متغیر | x = 5 | x=5 |
| += | افزودن به مقدار متغیر | x += 3 (معادل x = x + 3) | x=8 |
| -= | کم کردن از مقدار متغیر | x -= 3 (معادل x = x - 3) | x=2 |
| *= | ضرب کردن مقدار متغیر با یک عدد | x *= 3 (معادل x = x * 3) | x=15 |
| /= | تقسیم کردن مقدار متغیر با یک عدد | x /= 3 (معادل x = x / 3) | x=1.66 |
| //= | تقسیم صحیح (بدون اعشار) | x //= 3 (معادل x = x // 3) | x=1 |
| %= | محاسبه باقی‌مانده تقسیم | x %= 3 (معادل x = x % 3) | x=2 |
| **= | محاسبه توان | x **= 2 (معادل x = x ** 2) | x=25 |

### ۵- عملگرهای عضویت (Membership Operators) :

این عملگرها برای بررسی عضویت یک عنصر در یک رشته یا لیست استفاده میشوند .

| عملگر | توضیح | مثال | خروجی |
|---|---|---|---|
| in | بررسی اینکه آیا یک عنصر در داده‌ها وجود دارد یا خیر | 'a' in 'apple' | True |
| not in | بررسی اینکه آیا یک عنصر در داده‌ها وجود ندارد یا خیر | 'b' not in 'apple' | True |

### ۶- عملگرهای هویتی (Identity Operators) :

عملگرهای هویت برای بررسی اینکه آیا دو متغییر به یک شی اشاره میکنند یا نه، استفاده میشوند.

| عملگر | توضیح | مثال | خروجی |
|---|---|---|---|
| is | بررسی اینکه آیا دو متغیر به یک شیء اشاره می‌کنند یا خیر | x is y | True/False |
| is not | بررسی اینکه آیا دو متغیر به یک شیء اشاره نمی‌کنند | x is not y | True/False |

## اولویت اجرای عملگرها (Operator Precedence)

اولویت عملگرها مشخص میکند که در صورت وجود چندین عملگر در یک عبارت ، کدام یک ابتدا اجرا میشود. اولویت هادر پایتون از بالا به پایین به شرح زیر هستند :

۱- پرانتزها : بالاترین اولویت را دارند .

۲- عملگرهای ریاضی : به ترتیب (** و * و / و // و ٪ )

۳- عملگرهای مقایسه ای : (=> و =< و > و < و =! و == )

۴- عملگرهای منطقی : or , and , not

۵- عملگرهای انتساب : =** و =٪ و =// و =/ و =* و =- و =+ و =

### مقدار دهی متغییر ها و دستور نویسی پایتون :

```python
x = "Rahkar"
y = "is"
z = "your way"
print(x, y, z)
print(x + y + z)
x = 5
y = "Rahkar"
print(x)
print(y)
x = 1
y = 2.8
z = 1j
print(type(x))
print(type(y))
print(type(z))
```



```
Rahkar is your way
Rahkarisyour way
5
Rahkar
<class 'int'>
<class 'float'>
<class 'complex'>
```

```python
x = 35e3
y = 12E4
z = -87.7e100
print((x))
print((y))
print((z))
print('----')
print(type(x))
print(type(y))
print(type(z))
```



```
35000.0
120000.0
-8.77e+101
----
<class 'float'>
<class 'float'>
<class 'float'>
```

```python
x = 3+5j
y = 5j
z = -5j
print(x)
print(y)
print(z)
print(type(x))
print(type(y))
print(type(z))
```

```
(3+5j)
5j
(-0-5j)
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

عملکرد تابع int():

```python
x = int(1)
y = int(2.8)
z = int("3")
print(x)
print(y)
print(z)
```

```
1
2
3
```

عملکرد تابع float():

```python
x = float(1)
y = float(2.8)
z = float("3")
w = float("4.2")
print(x)
print(y)
print(z)
print(w)
```

```
1.0
2.8
3.0
4.2
```

عملکرد تابع str():

```python
x = str("s1")
y = str(2)
z = str(3.0)
print(x)
print(y)
print(z)
```

```
s1
2
3.0
```

# کار با رشته‌ها در پایتون

## فرمت‌بندی رشته‌ها با استفاده از f-strings در پایتون

در پایتون، **f-strings** یا **formatted string literals** یکی از بهترین و ساده‌ترین روش‌ها برای فرمت‌بندی رشته‌ها (strings)است. این قابلیت از پایتون 3.6 به بعد معرفی شد و به شما این امکان را می‌دهد که به راحتی مقادیر متغیرها را در داخل رشته‌ها قرار دهید بدون اینکه نیاز به استفاده از عملگرهای پیچیده یا توابع اضافی داشته باشید.

### نحوه استفاده از f-strings

برای استفاده از f-strings، کافی است که رشته را با حرف f قبل از علامت نقل‌قول (single یا double quotes) شروع کنید. سپس می‌توانید داخل رشته، از **آکولادها** {} برای قرار دادن متغیرها یا عبارات استفاده کنید.

### فرمت کلی:

```
f"متن ثابت {متغیر یا عبارت}"
```

### مثال‌های ساده از f-strings

1. **استفاده از متغیر در f-string:**

فرض کنید دو متغیر داریم .برای وارد کردن این متغیرها داخل یک رشته، می‌توانیم از f-string به‌صورت زیر استفاده کنیم:

```
name = "Ali"
age = 25
text = f"سلام {name}, شما {age} ساله هستید."
print(text)
```

پاسخ:

```
سلام Ali, شما 25 ساله هستید.
```

## فرمت‌بندی اعداد در f-strings

یکی از ویژگی‌های جالب f-strings این است که می‌توانید برای فرمت‌بندی اعداد، تاریخ‌ها و مقادیر دیگر از قالب‌های مختلف استفاده کنید.

**مثال 1: فرمت‌بندی اعداد اعشاری**

```
pi = 3.141592653589793
formatted = f"عدد پی برابر است با {pi:.2f}"
print(formatted)
```

**مثال 2: استفاده از فرمت برای اعداد بزرگ**

```
large_number = 1000000
formatted = f"عدد به‌صورت فرمت هزارگان: {large_number:,}"
print(formatted)
```

**مثال 3: فرمت تاریخ‌ها**

```
from datetime import datetime
current_date = datetime.now()
formatted = f"تاریخ امروز: {current_date:%Y-%m-%d %H:%M:%S}"
print(formatted)
```

# متدهای نوع داده رشته ای (String)

| متد | توضیح |
|---|---|
| endswith() | بررسی می‌کند که آیا رشته با یک زیررشته خاص تمام می‌شود. |
| count() | تعداد وقوع یک زیررشته در رشته را می‌شمارد. |
| zfill() | صفرهای اضافی به ابتدای رشته اضافه می‌کند تا طول رشته به اندازه مشخصی برسد. |
| expandtabs() | تمام کاراکترهای تب (\t) را به فضای سفید تبدیل می‌کند. |
| strip() | فضای خالی (whitespace) را از ابتدای و انتهای رشته حذف می‌کند. |
| isupper() | بررسی می‌کند که آیا تمام حروف رشته به حروف بزرگ است. |
| islower() | بررسی می‌کند که آیا تمام حروف رشته به حروف کوچک است. |
| isalnum() | بررسی می‌کند که آیا رشته فقط شامل حروف و اعداد است. |
| isnumeric() | بررسی می‌کند که آیا رشته تنها شامل اعداد است. |
| istitle() | بررسی می‌کند که آیا رشته مانند یک عنوان (با حروف بزرگ شروع شده و سایر حروف کوچک) است. |
| isidentifier() | بررسی می‌کند که آیا رشته یک شناسه معتبر پایتون است (مثل نام متغیر). |
| format() | فرمت‌بندی رشته‌ها با استفاده از جایگذاری مقادیر. |

| متد | توضیح |
|---|---|
| upper() | تمام حروف رشته را به حروف بزرگ تبدیل می‌کند. |
| lower() | تمام حروف رشته را به حروف کوچک تبدیل می‌کند. |
| capitalize() | اولین حرف رشته را بزرگ کرده و بقیه را کوچک می‌کند. |
| title() | اولین حرف هر کلمه را بزرگ می‌کند. |
| strip() | فاصله‌های اضافی در ابتدا و انتهای رشته را حذف می‌کند. |
| lstrip() | فاصله‌های اضافی از ابتدای رشته را حذف می‌کند. |
| rstrip() | فاصله‌های اضافی از انتهای رشته را حذف می‌کند. |
| find() | موقعیت اولین وقوع یک زیررشته را می‌دهد. اگر پیدا نشد -1 باز می‌گرداند. |
| replace() | یک زیررشته را با زیررشته‌ای دیگر جایگزین می‌کند. |
| split() | رشته را بر اساس یک جداکننده مشخص تقسیم می‌کند و یک لیست برمی‌گرداند. |
| join() | یک لیست از رشته‌ها را با جداکننده‌ای خاص به هم می‌چسباند. |
| isalpha() | بررسی می‌کند که آیا رشته تنها شامل حروف است. |
| isdigit() | بررسی می‌کند که آیا رشته تنها شامل اعداد است. |
| isspace() | بررسی می‌کند که آیا رشته تنها شامل فضای خالی است. |
| startswith() | بررسی می‌کند که آیا رشته با یک زیررشته خاص شروع می‌شود. |

## مبحث Slicing رشت ها در پایتون :

Slicing یکی از قابلیت های قدرتمند در پایتون است که به شما این امکان را میدهد که بخشی از یک رشته (String) یا لیست (List) را استخراج کنید. این ویژگی در پایتون بسیار پرکاربرد است و به شما کمک میکند تا به راحتی از داده ها برش بزنید و قطعات دلخواه را بدست آورید.

## نحوه استفاده از Slicing در پایتون:

برای انجام Slicing در پایتون، باید از ترکیب ایندکس ها استفاده کنید. این فرمت کلی برای Slicing است :

```
string[start:end:step]
```

Start: ایندکس شروع از کدام موقعیت در رشته یا لیست شروع کنیم.

END: ایندکس پایان. درواقع ایندکسی که نمیخواهید در آن گنجانده شود(نقطه ای کهبایدد متوقف شوید)

Step: فاصله یا گام بین دو عنصر. به شما این امکان را میدهد که فقط از هر n امین کاراکتر استفاده کنید.

جزئیات پارامترهای Slicing :

- ایندکس شروع : این پارامتر ایندکس اولین کاراکتر را که میخواهید در نتیجه باشد، مشخص میکند.اگر Start را نگذارید، به طور پیش فرض از اولین کاراکتر رشته شروع میشود(ایندکس0).
- End: (ایندکس پایان) این پارامتر ایندکس آخرین کاراکتری که میخواهید شامل نتیجه باشد را مشخص میکند. این ایندکس شامل نمیشود(به عبارت دیگر،از ایندکس پایان به بعد هیچ چیزی شامل نمیشود).
- ایندکس گام : این پارامتر به شما اجازه میدهد که هر n امین کاراکتر را انتخاب کنید.اگر آن را نگذارید ، به طور پیش فرض step=1 است(یعنی همه کاراکترها به ترتیب گرفته میشوند).

## مثال های کاربردی Slicing در پایتون

مثال ۱ : استخراج یک بخش از رشته فرض کنید رشته ی زیر را داریم:

```
text = "Hello, Rahkar!"
```

دریافت اولین ۵ کاراکتر:

```
print(text[:5])  # خروجی: 'Hello'
```

اینجا Start به طور پیش فرض 0 است و از ابتدای رشته تا ایندکس ۵ (غیر از آن) میرویم.

### دریافت همه کاراکترها از ایندکس ۷ تا ۱۲ :

```
print(text[7:12])  # خروجی: 'Rahka'
```

چون end را ننوشتیم ، پایتون به طور پیش فرض تا اخر رشته ادامه میدهد.

```
print(text[::2])  # خروجی: 'Hlo hk!'
```

دریافت همه کاراکترها با گام ۲(یعنی از هر ۲ امین کاراکتر).

### مثال۲: استفاده از پارامتر Step

گام معکوس(معکوس کردن رشته):

```
print(text[::-1])  # خروجی: '!rakahR ,olleH'
```

با استفاده از Step=-1 رشته به طور معکوس برگردانده میشود.

**انتخاب کاراکترها با گام ۳:**

```
print(text[::3])  # خروجی: 'Hl ak!'
```

در اینجا، از هر ۳ امین کاراکتر یک کاراکتر انتخاب میشود.

سایر نکات مهم در Slicing :

- استفاده از ایندکس منفی: در پایتون میتوانید از ایندکس منفی برای اشاره به کاراکترها از انتهای رشته استفاده کنید.ایندکس منفی به این صورت عمل میکند:
- ایندکس۱- آخرین کاراکتر رشته است.
- ایندکس ۲- دومین کاراکتر از انتها است و به همین ترتیب .

```
text='Hello Rahkar!'

print(text[-1])   # خروجی: '!'
print(text[-5:])  # خروجی: 'Rahkar!'
```

- استفاده از None در: Slicing شما میتوانید به طور پیش فرض end و start و step را با none مشخص کنید.
- None = Start به معنای شروع از اولین کاراکتر است.
- None = end به معنای رفتن تا انتهای رشته است.
- None = step به معنای استفاده از گام پیش فرض (۱) است.

```
print(text[None:None:None])  # همان رشته اصلی 'Hello, Rahkar!'
```

**جمع بندی**

- Slicingدر پایتون یک روش قدرتمنهد و مفید برای استخراج بخش هایی از یک رشته یا لیست است.
- اگر Start و End را نگذارید ، به طور پیش فرض از ابتدا تا انتها و با گام ۱ در نظر گرفته میشود.
- ایندکس منفی به شما این امکان را میدهد که از انتهای رشته به جلو حرکت کنید.
- برای تغییر گام ، میتوانید از پارامتر Step استفاده کنید.

## کار با لیست ها در پایتون :

لیست ها(List) د پایتون یکی از مهم ترین و پرکاربردترین ساختارهای داده هستند که برای ذخیره مجموعه ای از آیتم ها یاذعناصر به صورت ترتیب دار (Ordered) استفاده میشوند. لیست ها میتوانند شامل انواع مختلف داده ها باشند و تغییرپذیر هستند(یعنی میتوانند بعد از ساخت،تغییر کنند).

ویژگی های لیست ها :

- ترتیب دار(ordered): ترتیب عناصر در لیست حفظ میشود.
- تغییرپذیر(Mutable): میتوانیم عناصر یک لیست را تغییر دهیم.
- اجزای مختلف: لیست ها میتوانند شامل هر نوع داده ای (اعداد، رشته ها، لیست های دیگر، اشیاء و ...)باشند.
- دسترسی به عناصر: با استفاده از ایندکس میتوانیم به عناصر لیست دسترسی پیداکنیم.
- ایندکس منفی :میتوان از ایندکس های منفی برای دسترسی به عناصر از انتهای لیست استفاده کرد .

## تعریف لیست خالی:

```
a=list()
b=[]
```

```
a = ["apple", "banana". "cherry"]
print(a)
```

```
['apple', 'banana', 'cherry']
```

## دسترسی به آیتم‌های لیست

```
a = ["apple", "banana", "cherry"]
print(a[1])
```

```
banana
```

## ایجاد لیست تو در تو

```
a = ["apple", "banana", ["cherry", "blackcurrant"], "orange"]
print(a)
print(a[2])
print(a[2][1])
```

```
['apple', 'banana', ['cherry', 'blackcurrant'], 'orange']
['cherry', 'blackcurrant']
blackcurrant
```

## تغییر مقدار یکی از آیتم‌های لیست

```
a = ["apple", "banana", "cherry"]
a[1] = "blackcurrant"
print(a)
```

```
['apple', 'blackcurrant', 'cherry']
```

## چاپ اعضای لیست با حلقه for

```
mylist = ["apple", "banana", "cherry"]
for x in mylist:
    print(x)
```

```
apple
banana
cherry
```

# متدهای نوع داده List

| متد | توضیح | مثال |
|---|---|---|
| append() | .افزودن یک عنصر به انتهای لیست | my_list = [1, 2, 3]<br>my_list.append(4); print(my_list)<br># خروجی: [1, 2, 3, 4] |
| insert() | افزودن یک عنصر در موقعیت .مشخص شده (ایندکس) در لیست | my_list = [1, 2, 3]<br>my_list.insert(1, 'a')<br>print(my_list)<br># خروجی: [1, 'a', 2, 3] |
| remove() | حذف اولین وقوع یک عنصر از .لیست | my_list = [1, 2, 3, 2]<br>my_list.remove(2)<br>print(my_list)<br># خروجی: [1, 3, 2] |
| pop() | حذف و برگرداندن عنصر از انتهای .لیست (یا ایندکس خاص) | my_list = [1, 2, 3]<br>x = my_list.pop()<br>print(x)<br># خروجی: 3<br><br>my_list.pop(0)<br>print(my_list)<br># خروجی: [2, 3] |
| clear() | .حذف تمام عناصر لیست | my_list = [1, 2, 3]<br>my_list.clear()<br>print(my_list)<br># خروجی: [] |
| index() | یافتن ایندکس اولین وقوع یک .عنصر در لیست | my_list = [1, 2, 3]<br>print(my_list.index(2))<br># خروجی: 1 |
| count() | شمارش تعداد وقوع یک عنصر در .لیست | my_list = [1, 2, 2, 3]<br>print(my_list.count(2))<br># خروجی: 2 |

| متد | توضیح | مثال |
|---|---|---|
| sort() | مرتب‌سازی لیست (به‌طور پیش‌فرض به ترتیب صعودی). | my_list = [3, 1, 2]<br>my_list.sort(); print(my_list)<br># خروجی: [1, 2, 3] |
| reverse() | معکوس کردن ترتیب عناصر لیست. | my_list = [1, 2, 3]<br>my_list.reverse()<br>print(my_list)<br># خروجی: [3, 2, 1] |
| copy() | ساخت یک کپی از لیست. | my_list = [1, 2, 3]<br>new_list = my_list.copy()<br>print(new_list)<br># خروجی: [1, 2, 3] |
| extend() | افزودن تمام عناصر یک لیست به لیست دیگر. | list1 = [1, 2]<br>list2 = [3, 4]<br>list1.extend(list2)<br>print(list1)<br># خروجی: [1, 2, 3, 4] |
| pop() | حذف و برگرداندن عنصر از لیست (با ایندکس دلخواه). | my_list = [1, 2, 3]<br>print(my_list.pop(0))<br># خروجی: 1<br><br>print(my_list)<br># خروجی: [2, 3] |
| sort(reverse=True) | مرتب‌سازی لیست به صورت نزولی (یا معکوس). | my_list = [1, 3, 2]<br>my_list.sort(reverse=True)<br>print(my_list)<br># خروجی: [3, 2, 1] |
| reverse() | معکوس کردن لیست. | my_list = [1, 2, 3]<br>my_list.reverse()<br>print(my_list)<br># خروجی: [3, 2, 1] |

## توضیحات تکمیلی :

۱. () Append: فقط یک عنصر به انتهای لیست اضافه میکند.

۲. () Insert: عنصر را در ایندکس دلخواه در لیست اضافه میکند.

۳. () Remove: اولین وقوع از عنصر داده شده را از لیست حذف میکند.

۴. Pop (): یک عنصر را از انتهای لیست (یا ایندکس خاص) حذف کرده و برمیگرداند.
۵. Clear (): تمام عنصر لیست را حذف میکند.
۶. Index (): ایندکس اولین وقوع عنصر مورد نظر در لیست را برمیگرداند.

## کار با تاپل ها در پایتون:

تاپل(Tuple) در پایتون یکی از ساختارهای داده ای است که برای ذخیره مجموعه ای از داده ها استفاده میشود.مشابه لیست ها(list)اما با ویژگی های خاص خود که آن ها را از لیست ها متمایز میکند.

ویژگی های تاپل:

- غیر قابل تغییر(Immutable): برخلاف لیست ها، تاپل ها تغییرناپذیر هستند. به این معنی ک پس از تعریف یک تاپل ،نمیتوانیم عناصر آن را تغییر دهیم، اضافه کنیم یا حذف کنیم.
- ترتیب دار(Ordered): تاپل ها مانند لیست ها ترتیب دارند . به این معناست که میتوانیم به هر عنصر دسترسی پیدا کرده و از ایندکس ها برای مراجعه به آنها استفاده کنیم.
- دسترسی به عناصر از طریق ایندکس: مانند لیست ها ، میتوانیم از ایندکس برای دسترسی به عناصر تاپل استفاده کنیم.
- محتویات مختلف: تاپل میتواند از انواع داده های مختلف تشکیل شود(اعداد،رشته ها،لیست ها،و حتی دیگر تاپل ها).
- پشتیبانی از ایندکس منفی : همانند لیست ها ، میتوان از ایندکس های منفی برای دسترسی به عناصر از انتهای تاپل استفاده کرد.
- پشتیبانی از عملیات هایی مثل طول سنجی ، تکرار، تصال : میتوانید از عملیات هایی مانند Len()،+،* برای انجام کارهای مختلف روی تاپل ها استفاده کنید.
- تاپل ها با استفاده از () تعریف میشوند ، برخلاف لیست ها که با تعریف میشوند.

ایجاد تاپل خالی:

```
t1 = ()
t2 = tuple()
t3 = (5,)  # تعریف تاپل با یک عنصر
```

مقداردهی تاپل:

```
t1 = ("apple", "banana", "cherry")
print(t1)
```

```
('apple', 'banana', 'cherry')
```

دسترسی به آیتم های تاپل:

```
t1 = ("apple", "banana", "cherry")
print(t1[1])
```

```
banana
```

تغییر مقدار یکی از ایندکس های تاپل:

```
t1 = ("apple", "banana", "cherry")
t1 [1] = "blackcurrant"
# مقادیر در تاپل، قابل تغییر نیستند
print(t1)
```

نمایش خطا بدلیل عدم امکان تغییر آیتم‌های تاپل

```
Traceback (most recent call last):
  File "./prog.py", line 2, in <module>
TypeError: 'tuple' object does not support item assign
```

دسترسی به آیتم های تاپل با حلقه for :

```
t1 = ("apple", "banana", "cherry")
for x in t1:
      print(x)
```

```
apple
banana
cherry
```

کنترل موجود بودن یک مقدار داخل مقادیر تاپل:

```
t1 = ("apple", "banana", "cherry")
if "apple" in t1:
    print("Yes, 'apple' is in the fruits tuple")
```

```
Yes, 'apple' is in the fruits tuple
```

محاسبه طول تایپ:

```
t1 = ("apple", "banana", "cherry")
print(len(t1))
```

```
3
```

حذف کرن تاپل:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple)
```

این دستور موجب خطا میشه. چون تاپل دیگه وجود نداره

```
Traceback (most recent call last):
  File "demo_tuple_del.py", line 3, in <module>
    print(thistuple) #this will raise an error because the tuple no longer exists
NameError: name 'thistuple' is not defined
```

## اتصال (Concatenation) و تکرار(Repetition)

- می‌توانید دو تاپل را با استفاده از عملگر +ترکیب کنید.
- می‌توانید یک تاپل را چند بار تکرار کنید با استفاده از علامت *

```
tuple1 = (1, 2)
tuple2 = (3, 4)
combined = tuple1 + tuple2
print(combined)  # خروجی: (1, 2, 3, 4)

repeated = tuple1 * 3
print(repeated)  # خروجی: (1, 2, 1, 2, 1, 2)
```

## تعداد و وجود(Count & Index)

- `count(x)` تعداد دفعاتی که xدر تاپل آمده است را برمی‌گرداند.
- `index(x)` ایندکس اولین وقوع xرا در تاپل می‌دهد .

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

## مزایای استفاده از تاپل‌ها:

- **عملکرد سریع‌تر نسبت به لیست‌ها :**به دلیل تغییرناپذیری، تاپل‌ها معمولاً سریع‌تر از لیست‌ها هستند.
- **اطمینان از عدم تغییر :**در برخی مواقع، زمانی که نیاز دارید داده‌ها تغییر نکنند، از تاپل استفاده می‌شود.
- **قابلیت استفاده به‌عنوان کلید دیکشنری :**از آنجا که تاپل‌ها غیر قابل تغییر هستند، می‌توانند به‌عنوان **کلید در دیکشنری‌ها** استفاده شوند، در حالی که لیست‌ها نمی‌توانند چنین ویژگی‌ای داشته باشند.

ایجاد مجموعه:

```
s1 = set()
s2={4 , 1 , 32 , 15}
s3={15 , 0 , 26 , (11 , 5 , 96) , 80 }
print(s1)
print(s2)
print(s3)
```

```
set()
{32, 1, 4, 15}
{0, 15, 80, (11, 5, 96), 26}
```

توجه: لیست مجموعه نامرتب است، به این معنی: موارد به ترتیب تصادفی ظاهر می شوند.

دسترسی به اعضای مجموعه با حلقه for :

```
s1 = {"apple", "banana", "cherry"}
for x in s1:
    print(x)
```

```
apple
cherry
banana
```

کنترل موجود بودن یک مقدار در مجموعه :

```
s1 = {"apple", "banana", "cherry"}
s1.add("orange")
print(s1)
```

```
{'apple', 'banana', 'orange', 'cherry'}
```

استخراج طول مجموعه

:

```
s1 = {"apple", "banana", "cherry"}
print(len(s1))
```

```
3
```

حذف مقدار ازمتد discard() :

```
s1 = {"apple", "banana", "cherry"}

s1.discard("banana")

print(s1)
```

```
{'apple', 'cherry'}
```

حذف اخرین آیتم با متد pop() :

```
s1 = {"apple", "banana", "cherry"}
x = s1.pop()
print(x) # حذف آیتم
print(s1) # مجموعه بعد از حذف آیتم فوق
```

```
banana
{'apple', 'cherry'}
```

خالی کردن محتوای مجموعه:

```
s1 = {"apple", "banana", "cherry"}
s1.clear()
print(s1)
```

```
set()
```

**تفاضل :(Difference)** برای پیدا کردن اعضای یک مجموعه که در مجموعه دیگر نیستند از عملگر -یا متد `difference()`استفاده می‌شود.

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
result = set1 - set2
print(result)  # خروجی: {2 ,1}
```