# ساختمان داده‌ها با پایتون


**تألیف:**

**دکتر جواد وحیدی**
**دکتر رمضان عباس نژاد ورزی**



**فن‌آوری نوین**

| | |
|---|---|
| سرشناسه | وحید- جواد، |
| عنوان و نام پدیدآور | ساختمان داده‌ها با پایتون/ تألیف رمضان عباس، جواد وحیدی |
| مشخصات نشر | بابل:فن‌آوری نوین, ۱۳۹۷ |
| مشخصات ظاهری | ۳۱۲ ص :مصور،جدول |
| شابک | ۴۱۵۰۰۰ ریال :۸-۲۶-۷۲۷۲-۶۰۰-۹۷۸ |
| وضعیت فهرست نویسی | فیپا |
| موضوع | پیتون (زبان برنامه‌نویسی کامپیوتر) |
| موضوع | (Python ) Computer program language |
| شناسه افزوده | عباس‌نژاد ورزی، رمضان، ۱۳۴۸ - |
| رده‌بندی کنگره | ۱۳۹۷/پ۹۵۷۳/۷۶QA ۳و |
| رده‌بندی دیویی | ۰۰۵/۱۳۳ |
| شماره کتابشناسی ملی | ۵۵۲۰۰۰۴ |

www.fanavarienovin.net

**فن‌آوری نوین  بابل، کد پستی ۴۷۱۶۷-۷۳۴۴۸  تلفن: ۳۲۲۵۶۶۸۷-۰۱۱**

## ساختمان داده‌ها با پایتون

**تألیف:** جواد وحیدی - رمضان عباس نژادورزی

**نوبت چاپ:** چاپ اول

**سال چاپ:** زمستان ۱۳۹۷

**شمارگان:** ۱۰۰۰

**قیمت:** ۴۱۵۰۰ تومان

**نام چاپخانه و صحافی:** دفتر فنی سورنا

**شابک:** ۸- ۲۶-۷۲۷۲-۶۰۰-۹۷۸

**نشانی ناشر: بابل، چهارراه نواب، کاظم بیگی، جنب مسجد منصور کاظم بیگی، طبقه اول**

**طراح جلد:** کانون آگهی و تبلیغات آبان (احمد فرجی)

**تهران، خ اردیبهشت، نبش وحید نظری، پلاک ۱۴۲  تلفکس: ۶۶۴۰۰۱۴۴-۶۶۴۰۰۲۲۰**

# فهرست مطالب

# مقدمه

ساختمان داده یکی از درس‌های پایه‌ای و مهم رشته مهندسی کامپیوتر، فناوری اطلاعات و علوم کامپیوتر است. کتاب‌های زیادی در زمینه ساختمان داده‌ها تألیف و ترجمه‌شده است که جای تشکر دارد. کتاب‌های موجود، روی یک بخش خاص متمرکز شده‌اند. اما، کتاب حاضر علاوه بر تدریس مفاهیم ساختمان داده‌ها با مثال‌های متعدد، الگوریتم‌های بیان‌شده را با زبان پایتون پیاده‌سازی نموده است. از نکات بارز این کتاب علاوه بر تدریس علمی مفاهیم نکات تستی را نیز بیان نموده است. در همین راستا حدود ۴۵۰ تست کنکور کارشناسی ارشد رشته‌های مهندسی کامپیوتر، فناوری اطلاعات و علوم کامپیوتر دانشگاه‌های دولتی و آزاد به همراه حل تشریحی آن‌ها در پیوست کتاب الکترونیکی آمده است.

این کتاب شامل ۷ فصل است. در فصل اول، الگوریتم‌ها، پیچیدگی آن‌ها و الگوریتم‌های بازگشتی بیان گردیده است. در فصل دوم، کتاب آرایه، ماتریس، ماتریس خلوت و کاربردهای آن‌ها آمده است. در فصل سوم، مفاهیم صف و پشته و کاربردآن‌ها شرح داده‌شده است. در فصل چهارم، لیست پیوندی بیان گردیده است. در فصل پنجم، درخت و کاربردهای آن را می‌بینید. در فصل ششم، گراف و کاربردهای آن بحث شده است و در خاتمه در فصل هفتم، روش‌های مختلف مرتب‌سازی بیان گردیده است.

علاوه بر فصل‌های بیان‌شده به کتاب الکترونیکی دو پیوست اضافه گردیده است. پیوست اول به آموزش برنامه‌نویسی پایتون به‌طور خلاصه پرداخته است. اما، پیوست ۴۵۰ تست کارشناسی ارشد را مطرح نموده و به‌صورت تشریحی حل نموده است.

امیدواریم این اثر نیز موردتوجه اساتید و دانشجویان عزیز واقع شود.

پیاده‌سازی‌های برنامه‌ها را می‌توانید از سایت انتشارات فن‌آوری نوین به آدرس www.fanavarienovin.net دریافت نمایید.

در پایان از تمامی خوانندگان عزیز (اساتید و دانشجویان) تقاضا داریم، هرگونه اشکال، ابهام در متن کتاب، پیشنهادات و انتقادات را به آدرس پست الکترونیک fanavarienovin@gmail.com ارسال نمایند.

بابل، زمستان ۱۳۹۷

مؤلفین

| لیست دیگر کتاب‌های انتشارات |
| --- |
| گنجینه سوالات C# (حل مسائل-مرجع کامل) |
| مفاهیم شی‌گرایی و پیاده‌سازی آن‌ها با زبان‌های C# ، جاوا، C++ و پایتون |
| مسئله مسیریابی وسایل نقلیه (تئوری و کاربردها) |
| روسازی راه |
| آموزش گام‌به‌گام برنامه نویسی C#- From Application |
| آموزش گام‌به‌گام برنامه‌نویسی پایتون |
| حل مسائل پایتون (حل ۶۵۰برنامه -مرجع کامل) |
| مبانی رایانه و برنامه‌نویسی به زبان C++ |
| 600 برنامه C++ با حل کامل آنها(حل مسائل- C++ مرجع کامل) |
| حل مسائل جاوا (حل ۶۰۰ برنامه - مرجع کامل) |
| طراحی سیستم‌های شی‌گرا با زبان C # |
| 650 برنامه C # با حل کامل آنها(حل مسائل #- مرجع ) |
| دانلود کتاب آموزش گام‌به‌گام برنامه‌نویسی بانک اطلاعاتی با C# |
| دانلود کتاب حل مسائل C# |
| گرافیک رایانه‌ای با زبان برنامه‌نویسی C# |
| آشنایی با مبانی امنیت شبکه (امنیت اطلاعات) |
| اصول طراحی پایگاه داده‌ها |
| حل مسائل C++ (آزمایشگاه کامپیوتر مرجع کامل) |
| آموزش گام‌به‌گام LINQ با C# |
| ساختمان داده‌ها با C++ |
| مدیریت استراتژیک فناوری اطلاعات |
| درس و کنکور پایگاه داده پیشرفته |
| فیزیک الکتریسیته |
| تجارت الکترونیکی |
| راهنمای کاربردی کاربری OPNET برای شبکه‌های شبیه‌سازی کامپیوتر |
| درس و کنکور سیستم عامل پیشرفته |
| شبکه‌های کامپیوتری با رویکرد کاربردی، آزمایشگاه شبیه‌سازی شبکه |

| |
|---|
| آزمایشگاه پایگاه داده با SQL Server 2012 |
| کاربرد رایانه در مدیریت و حسابداری |
| آموزش گام‌به‌گام برنامه‌نویسی بانک اطلاعاتی با ویژوال بیسیک‌نت |
| آموزش گام‌به‌گام برنامه‌نویسی به زبان C++ |
| حل مسائل پاسکال |
| حل مسائل C++ |
| دانش ثروتمند شدن |
| برنامه سازی پیشرفته به زبان C# |
| الگوریتم ها و محاسبات موازی |
| دانلود کتاب حل مسائل C |
| قدرت فکر |
| ارتعاش فکر |
| طراحی سیستم‌های تحمل‌پذیر خطا |
| طراحی واسط کاربری با پایتون |
| حل مسائل ساختمان داده با پایتون |

| نام کتاب | لینک خرید چاپی | لینک خرید الکترونیکی | لینک فایل نمونه |
|---|---|---|---|
| گنجینه سوالات#C (حل مسائل–مرجع کامل) | *لینک خرید کتاب چاپی* | لینک خرید فایل الکترونیکی | |
| مفاهیم شی‌گرایی و پیاده‌سازی آن‌ها با زبان‌های#C ، جاوا، C++ و پایتون | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| مسئله مسیریابی وسایل نقلیه (تئوری و کاربردها) | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| روسازی راه | | لینک خرید فایل الکترونیکی | |
| آموزش گام‌به‌گام برنامه نویسی#C- From Application | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| آموزش گام‌به‌گام برنامه‌نویسی پایتون | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| حل مسائل پایتون (حل۶۵۰برنامه –مرجع کامل) | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| مبانی رایانه و برنامه‌نویسی به زبان C++ | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| 600 برنامه C++ با حل کامل آنها(حل مسائل- C++ مرجع کامل) | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| حل مسائل جاوا (حل ۶۰۰ برنامه – مرجع کامل) | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| طراحی سیستم‌های شی‌گرا با زبان C # | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| 650 برنامه C # با حل کامل آنها(حل مسائل #- مرجع) | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| دانلود کتاب آموزش گام‌به‌گام برنامه‌نویسی بانک اطلاعاتی با C# | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| دانلود کتاب حل مسائل C# | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| گرافیک رایانه‌ای با زبان برنامه‌نویسی#C | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| آشنایی با مبانی امنیت شبکه (امنیت اطلاعات) | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| اصول طراحی پایگاه داده‌ها | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| حل مسائل C++ (آزمایشگاه کامپیوتر مرجع کامل) | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| آموزش گام‌به‌گام LINQ با C# | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| ساختمان داده‌ها با++C | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| مدیریت استراتژیک فناوری اطلاعات | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| درس و کنکور پایگاه داده پیشرفته | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| فیزیک الکتریسیته | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| تجارت الکترونیکی | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| راهنمای کاربردی کاربری OPNET برای شبکه‌های شبیه‌سازی کامپیوتر | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| درس و کنکور سیستم عامل پیشرفته | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| شبکه‌های کامپیوتری با رویکرد کاربردی، آزمایشگاه شبیه‌سازی شبکه | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| آزمایشگاه پایگاه داده با SQL Server 2012 | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| کاربرد رایانه در مدیریت و حسابداری | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| آموزش گام‌به‌گام برنامه‌نویسی بانک اطلاعاتی با ویژوال بیسیک‌نت | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |

| | | | |
|---|---|---|---|
| آموزش گام‌به‌گام برنامه‌نویسی به زبان C++ | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| حل مسائل پاسکال | | لینک خرید فایل الکترونیکی | |
| حل مسائل C++ | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | لینک دانلود فایل نمونه |
| دانش ثروتمند شدن | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| برنامه سازی پیشرفته به زبان C# | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| الگوریتم ها و محاسبات موازی | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| دانلود کتاب حل مسائل C | *لینک خرید کتاب چاپی* | لینک خرید فایل الکترونیکی | |
| قدرت فکر | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| ارتعاش فکر | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |
| طراحی سیستم‌های تحمل‌پذیر خطا | لینک خرید کتاب چاپی | لینک خرید فایل الکترونیکی | |

# فصل ۱ ساختار داده‌ها، الگوریتم‌ها و پیچیدگی

عوامل زیادی در سرعت اجرای برنامه مؤثرند. دو عامل بسیار مهم کارایی برنامه عبارت‌اند از:

۱. **ساختمان داده در نظر گرفته‌شده برای برنامه**

۲. **الگوریتم.** برنامه‌ای که حافظه کم‌تری مصرف کند و از الگوریتم کاراتری (سریع‌تر) استفاده نماید، بهتر است.

## ۱-۱. ساختمان داده‌ها

هر برنامه از دو بخش بسیار مهم تشکیل می‌شود که عبارت‌اند از:

۱. ساختمان داده‌ها ۲. الگوریتم‌ها.

**ساختمان داده‌ها**، برای دریافت داده‌ها توسط رایانه جهت پیاده‌سازی و اجرای الگوریتم‌ها مورداستفاده قرار می‌گیرند. اما، **الگوریتم‌ها**، دستورالعمل‌هایی هستند که بر روی داده‌ها اعمال می‌شوند.

برنامه‌ای خوب است که ساختمان داده‌ای مناسب داشته و از الگوریتم‌های بهینه‌تری استفاده کند. به‌عنوان‌مثال، فرض کنید، بخواهید ۲۰ عدد صحیح را خوانده، مرتب کنید و نمایش دهید. در این مثال، بهترین ساختمان داده آرایه است. چون، تعداد اعداد ثابت (مشخص‌شده) است. اما، الگوریتم‌های مختلف مرتب‌سازی از قبیل Bubble، Select، Insert، Quick، Shell و غیره وجود دارند که در فصل ۷ به آن‌ها می‌پردازیم. در ادامه می‌آموزیم کدام‌یک از الگوریتم‌های مرتب‌سازی را انتخاب کنیم.

در این بخش به انواع ساختمان داده‌ها می‌پردازیم. انواع مختلف ساختمان داده‌ها وجود دارند که عبارت‌اند از.

### ۱-۱-۱. ساختمان داده‌های ایستا

این نوع ساختمان داده‌ها، تعداد عناصرشان ثابت (ایستا) است. یعنی، در هنگام کامپایل (ترجمه) باید تعداد عناصر آن مشخص باشد. ساختمان داده‌های ایستا نیز دو نوع‌اند که عبارت‌اند از:

✪ **ساختمان داده‌های اولیه**، ساختمان داده‌هایی از قبیل داده‌های عددی صحیح (int، long، unsigned int و ...)، عددی اعشاری (float و double)، کاراکتری (char)، منطقی (boolean) و اشاره‌گر، ساختمان داده اولیه هستند.

✪ **ساختمان داده‌های ترکیبی (غیر اولیه)**، از ترکیبی از داده‌های ساده تشکیل می‌شوند. برخی از این ساختمان داده‌ها در زیر آمده‌اند:

### آرایه

تعدادی از خانه‌های پشت سر هم حافظه که دارای یک نام و یک نوع باشند. برای تعریف آرایه می‌توانید از کلاس array استفاده کنید. این کلاس در ماژول array قرار دارد. لذا، برای استفاده از این کلاس ابتدا باید ماژول array را با دستور زیر به برنامه اضافه کنید:

```
from array import *
```

اکنون می‌توانید از کلاس آرایه به‌صورت زیر استفاده کنید:

```
array(typecode [, initializer]) -> array
```

این کلاس دو پارامتر را دریافت می‌کند. پارامتر typecode، نوع آرایه را مشخص می‌کند و پارامتر [initalizer]، مقادیر اولیه آرایه را تعیین می‌کند. در فصل دوم با آرایه و کاربرد آن آشنا خواهیم شد.

### رشته

یکی از پرکاربردترین انواع داده در پایتون است. رشته‌ها، دنباله‌ای از کاراکترها هستند که در داخل تک کتیشن(') یا جفت کتیشن(") قرار می‌گیرند. به‌عنوان‌مثال، دستوارت زیر را ببینید:

```
>>> URL = 'WWW.FANAVARIENOVIN.NET'
>>> LANGUAGE = "PYTHON PROGRAMMING"
```

دستور اول، رشته‌ای به نام url تعریف کرده، مقدار ` WWW.FANAVARIENOVIN.NET ` را در آن قرار می‌دهد، دستور دوم، متغیری به نام language با مقدار "PYTHON PROGRAMMING" ایجاد می‌کند

### لیست

یکی از انواع آماده دیگر در پایتون نوع **لیست**[1] است. لیست در پایتون مجموعه‌ای از مقادیر هستند که در یک متغیر قرار می‌گیرند. یعنی، لیست مانند رشته از دنباله‌ای از مقادیر تشکیل می‌شود، اما برخلاف رشته، یک نوع **تغییرپذیر**[2] است. هر عضو لیست می‌تواند هر نوع داده باشد. یعنی، اعضای لیست می‌توانند انواع مختلف داشته باشند. به‌عبارت‌دیگر، یک عضو رشته‌ای باشد و عضو دیگر عددی باشد. حتی گاهی اوقات اعضای آن می‌توانند از نوع لیست باشند. اعضای لیست با کاما (,) از یکدیگر جدا می‌شوند. برای تعریف لیست از عملگرهای [ ] استفاده می‌شود. به‌عنوان‌مثال، دستور زیر را ببینید:

```
>>> list1 = []
```

این دستور یک لیست خالی به‌نام list1 ایجاد می‌کند. به‌جای این دستور می‌توان از دستور زیر استفاده کرد:

```
>>> list1 = list()
```

اکنون دستورات زیر را ببینید:

```
>>> list1 = ["Fanavarienovin", 2, True]
>>> list1[0]
```

[1] List [2] Mutable

دستور اول، یک لیست به‌نام list1 ایجاد می‌کند و به اعضای آن را به ترتیب "Fanavarienovin"، 2 و True تخصیص می‌دهد، دستور دوم، محتوی عضو اول آن (list1[0]) یعنی، همان `Fanavarienovin` را نمایش می‌دهد.

اندیس لیست از صفر شروع می‌شود و با استفاده از اندیس می‌توان به اعضای لیست دسترسی یافت. علاوه بر اندیس با استفاده از عملگر slice می‌توان به بخشی از لیست دسترسی یافت. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> list1 = [1, "Python", "Program language", True, [3, "Ali"]]
>>> list1[2:4]
```

دستور اول، لیستی به نام list1 با پنج عضو تعریف کرده، مقادیری را به آن‌ها تخصیص می‌دهد، دستور دوم، اعضای سوم و چهارم list1 را نمایش می‌دهد (خروجی زیر):

```
['Program language', True']
```

**تاپل (چندتایی)**

چندتایی‌ها[1] مشابه لیست‌ها هستند، با این تفاوت که چندتایی‌ها تغییرناپذیرند. یعنی، نمی‌توانید چندتایی‌ها را تغییر دهید. اعضای چندتایی با کاما ( , ) از یک‌دیگر جدا می‌شوند. چندتایی‌ها برای مواقعی استفاده می‌شوند که یک دستور یا تابعی که توسط کاربر تعریف‌شده است به مجموعه‌ای از مقادیر ثابت نیاز داشته باشد. در هنگام استفاده و تعریف چندتایی به نکات زیر دقت کنید:

۱. اعضای چندتایی با کاما از هم جدا می‌شوند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> tuple='f','a','n','a','v','a','r','i','e','n', 'o', 'v', 'i', 'n'
>>> tuple
```

دستور اول، یک چندتایی به‌نام tuple تعریف می‌کند و حروف `f`، `a`، `n`، `a`، `v`، `a`، `r`، `i`، `e`، `n`، `o`، `v`، `i` و `n` را به اعضای آن تخصیص می‌دهد و دستور دوم، محتوی چندتایی tuple را نمایش می‌دهد (خروجی زیر):

```
('f', 'a', 'n', 'a', 'v', 'a', 'r', 'i', 'e', 'n', 'o', 'v', 'i', 'n')
```

۲. در هنگام تعریف چندتایی مرسوم است که اعضای آن را در داخل () قرار گیرد، امّا ضروری نیست. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> insuranceType = (7, 9, 10)
>>> insuranceType
```

دستور اول، چندتایی به‌نام insuranceType (نوع بیمه) تعریف کرده و مقادیر ۷، ۹ و ۱۰ را به اعضای آن تخصیص می‌دهد، دستور دوم، اعضای چندتایی insuranceType را نمایش می‌دهد:

```
(7, 9, 10)
```

۳. چندتایی با صفر عضو نیز می‌توان ایجاد کرد. برای این منظور، به‌صورت زیر عمل می‌شود:

```
>>> emptyTuple = ()
```

[1] Tuple

این دستور، یک چندتایی به‌نام emptyTuple ایجاد می‌کند که هیچ عضوی ندارد. اکنون، اگر با تابع print() آن را چاپ کنید (یعنی، دستور زیر را تایپ کنید):

```
>>> print(emptyTuple)
```

خروجی به‌صورت ( ) نمایش داده می‌شود:

۴. برای ایجاد چندتایی با یک عضو، حتماً باید یک کاما در انتهای تعریف قبل از پرانتز بسته آورده شود. به‌عنوان‌مثال، دستورات زیر را مشاهده کنید:

```
>>> tuple1 = ('net', )
>>> tuple1
>>> type(tuple1)
```

دستور اول، چندتایی به‌نام tuple1 با یک عضو تعریف می‌کند، دستور دوم، محتوی اعضای چندتایی tuple1 (یعنی، مقدار ('net',) ) را نمایش می‌دهد و دستور سوم، نوع tuple1 را نمایش می‌دهد که نشان می‌دهد یک چندتایی است (خروجی زیر ):

```
<class 'tuple'>
```

امّا، اگر در هنگام تعریف چندتایی با یک عضو کاما ( , ) آخر را فراموش کنید، پایتون آن متغیر را به‌عنوان نوع رشته‌ای در نظر می‌گیرد. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> tuple1 = ('net')
>>> tuple1
>>> type(tuple1)
```

دستور اول، متغیری به‌نام tuple1 تعریف می‌کند و رشته `net` را در آن قرار می‌دهد. چون، کاما قبل از پرانتز قرار داده نشده است، پس پایتون آن را به‌عنوان رشته در نظر می‌گیرد، دستور دوم، محتوی tuple1 یعنی `net` را نمایش می‌دهد و دستور سوم، نوع tuple1 (یعنی، مقدار <class 'str'>) را نمایش می‌دهد که نشان می‌دهد یک کلاس str است.

۵. با عملگر اندیس می‌توان به عضوی از چندتایی دسترسی یافت. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> tuple1 = ('p', 'y', 't', 'h', 'o', 'n')
>>> tuple1[3]
```

با اجرای دستور اول، یک چندتایی به‌نام tuple1 ایجاد می‌شود، و اعضای آن را به ترتیب `p`، `y`، `t`، `h`، `o` و `n` تخصیص می‌دهد و دستور دوم، عضو با اندیس سه (یعنی، چهارمین عضو چندتایی) همان `h` را نمایش می‌دهد.

۶. با عملگر برش (slice) می‌توان محدوده‌ای از اعضای یک چندتایی را انتخاب نمود. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> tuple1 = ('A', 'B', 'C', 'D', 'E', 'F')
>>> tuple1[1:4]
```

دستور اول، چندتایی به‌نام tuple1 تعریف می‌کند و به اعضای آن مقادیر `A`، `B`، `C`، `D`، `E` و `F` تخصیص می‌دهد، دستور دوم، دومین تا چهارمین عضو چندتایی (مقادیر ('B', 'C', 'D')) را نمایش می‌دهد.

۷. اگر بخواهید عضوی از چندتایی را تغییر دهید، با خطا مواجه خواهید شد. به‌عنوان‌مثال، دستورات زیر را مشاهده کنید:

```
>>> tuple1 =('a', 'b', 3, 5)
>>> tuple1[1] ='c'
```

دستور اول، یک چندتایی به‌نام tuple1 تعریف کرده، به اعضای آن به ترتیب مقادیر `a`، `b`، 3 و 5 تخصیص می‌دهد، دستور دوم، می‌خواهد دومین عضو چندتایی را با مقدار 'c' تعویض کند که با خطای زیر مواجه می‌گردد:

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    tuple1[1] ='c'
TypeError: 'tuple' object does not support item assignment
```

۸. همان‌طور که بیان گردید، اعضای چندتایی را نمی‌توان تغییر داد، امّا، می‌توان یک چندتایی را با چندتایی جدید جایگزین کرد. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> tuple1 = ('a', 'b', 'd', 'e', 'f')
>>> tuple1 = tuple1[0:2] + ('c',) + tuple1[2:]
>>> print(tuple1)
```

دستور اول، یک چندتایی به‌نام tuple1 با اعضای `a`، `b`، `d`، `e` و `f` تعریف می‌کند، دستور دوم، این چندتایی تعریف‌شده tuple1 را با مقادیر زیر جایگزین کرده، دستور سوم، آن را نمایش می‌دهد :

( `a` , `b` , `c` , `d` , `e` , `f`)

tuple1[2:0] + (`c`,) + tuple1[2: ]

۹. نوع چندتایی به دلیل تغییرناپذیر بودن، نسبت به نوع لیست در مصرف حافظه بهینه‌تر است؛ بنابراین، بهتر است در مواقعی که نیاز به تغییر خاصی در داده‌ها نیست، از نوع چندتایی استفاده کنید.

۱۰. با تابع ()len می‌توان تعداد اعضای چندتایی را به دست آورد. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> tuple1 = (1, (2, 3))
>>> len(tuple1)
```

دستور اول، متغیری به‌نام tuple1 با نوع چندتایی ایجاد می‌کند که عضو اول آن مقدار ۱ و عضو دوم آن یک چندتایی با مقادیر (2, 3) دارد و دستور دوم، تعداد اعضای چندتایی tuple1 یعنی ۲ را نمایش می‌دهد.

۱۱. عملگرهای in، not in، ==، +، * را نیز می‌توان مانند لیست برای چندتایی‌ها استفاده نمود. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> tuple1 = (1, 2, 3)
>>> tuple2 = tuple1 + (4, 5, 6)
>>> 4 in tuple1
>>> 4 in tuple2
>>> 4 not in tuple1
>>> tuple1 == (1, 2, 3)
>>> tuple1 == tuple2
```

دستور اول، متغیری به‌نام tuple1 با نوع چندتایی و مقادیر (1, 2, 3) ایجاد می‌کند، دستور دوم، متغیری به‌نام tuple2 با نوع چندتایی و مقادیر (1, 2, 3, 4, 5, 6) ایجاد می‌کند، دستور سوم، چون ۴ در tuple1 وجود ندارد، False را نمایش می‌دهد، دستور چهارم، چون ۴ در tuple1 وجود دارد، True را نمایش می‌دهد، دستور پنجم، چون ۴ در tuple1 وجود ندارد و عملگر not in است، True را نمایش می‌دهد، دستور ششم، چون tuple1 برابر با تاپل (1, 2, 3) است، True را نمایش می‌دهد و دستور هفتم، چون tuple1 برابر با tuple2 نیست، False را نمایش می‌دهد.

**دیکشنری**

داده‌های رشته‌ای، لیست و چندتایی که تاکنون دیده‌اید، برای دسترسی به اعضای‌شان از اعداد صحیح به‌عنوان اندیس استفاده می‌کردند. در این نوع داده‌ها، اگر برای دسترسی به اعضای آن‌ها از اعداد غیر صحیح استفاده کنید، با خطا مواجه می‌شوید. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> list1 = [1, 3, 6, 8]
>>> print(list1[2])
```

دستور اول، لیستی به‌نام list1 با اعضای ۱، ۳، ۶ و ۸ ایجاد می‌کند و دستور دوم، سومین عضو list1 را نمایش می‌دهد (یعنی، ۶ را نمایش می‌دهد).

اکنون، دستور مقابل را اجرا کنید:

```
>>> print(list1[20.])
```

با اجرای این دستور مفسر پایتون خطای زیر را نمایش می‌دهد:

```
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    list1[2.0]
TypeError: list indices must be integers or slices, not float
```

به دلیل این‌که در اندیس از اعداد اعشاری استفاده گردید، با خطا مواجه شده است. زیرا، اندیس آرایه‌ها، رشته‌ها، چندتایی‌ها و لیست‌ها باید عدد صحیح باشد.

اما، دیکشنری داده‌های مرکبی هستند که هر نوع داده تغییرپذیری می‌تواند به‌عنوان اندیس آن استفاده شود. به‌عنوان‌مثال، اگر یک دیکشنری جهت ترجمه کلمات انگلیسی به فارسی استفاده شود، رشته، به‌عنوان اندیس این دیکشنری به کار می‌رود. برای این منظور، دیکشنری خالی ایجاد کرده، اعضا را به آن اضافه کنید. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>>pubTel={}
>>>pubTel['fanavarienovin'] = '01132256687'
>>> pubTel['daneshiran'] = '02166400220'
>>> pubTel['daneshnegar'] = '02166400144 '
>>> print(pubTel)
```

دستور اول، دیکشنری به‌نام pubTel ایجاد می‌کند که هیچ عضوی ندارد، دستورات دوم تا چهارم، سه عضو به این دیکشنری اضافه می‌کنند و دستور پنجم، محتوی دیکشنری را نمایش می‌دهد (خروجی زیر):

```
{'daneshiran': '02166400220', 'fanavarienovin': '01132256687', 'daneshnegar':
'02166400144 '}
```

همان‌طور که در خروجی تابع ()print مشاهده می‌شود، هر عضو دیکشنری شامل یک انـدیس و یـک مقدار است که به‌وسیله علامت : (کولن) از هم جدا می‌شوند. در یک دیکشنری اندیس‌هـا **کلیـد**[1] نامیـده می‌شوند. پس اعضای دیکشنری از جفت‌های **کلید – مقدار**[2] تشکیل می‌شوند.

اکنون دستور زیر را ببینید:

```
>>> pubTel['daneshnegar']
```

این دستور شـماره تلفـن انتشـارات دانـش نگـار (`daneshnegar`) یعنـی ' 02166400144' را نمـایش می‌دهد.

**تمام کلیدهای یک شیء دیکشنری باید یکتا[3] باشند.**

### مجموعه‌ها

همان‌طور که با مفهوم مجموعه در ریاضیات آشنا شدید، هر مجموعه از تعدادی عضو تشکیل می‌شود و هر عضو مجموعه باید یکتا (منحصربه‌فرد) باشد. در زبان پایتون به‌سادگی می‌توان مجموعه‌ای را تعریـف کرده و از آن استفاده نمود. برای این منظور، دو روش وجود دارد که عبارت‌اند از :

۱. **تعریف مجموعه با عملگرهـای** { }، اعضـای مجموعـه را مـی‌تـوان بـین { } قـرار داد. بـه‌عنـوان‌مثـال، دستورات زیر را ببینید:

```
>>> s1 = { 1, 2, 3, 4, 5, 6 }
>>> type(s1)
>>> print(s1)
```

دستور اول، مجموعه‌ای به نام $s_1$ با اعضای ۱ تا ۶ تعریف مـی‌کنـد، دسـتور دوم، نـوع $s_1$ (یعنـی، <class 'set'> ) را نمایش می‌دهد و دستور سوم، اعضای مجموعه $s_1$ را به‌صورت زیر نشان می‌دهد:

{1, 2, 3, 4, 5, 6}

۲. **تعریف مجموعه با کلاس set**، برای تعریف مجموعه می‌توان از کـلاس set بـه‌صـورت زیـر اسـتفاده کرد:

**نام متغیر مجموعه = set (لیست)**

به‌عنوان‌مثال، دستورات زیر را مشاهده کنید:

```
>>> l1 = [1, 2, 3, 4, 5, 6]
>>> s1 = set(l1)
>>> type(s1)
<class 'set'>
>>> print(s1)
```

دستور اول، لیستی به نام $l_1$ با مقادیر ۱ تا ۶ ایجاد می‌کند، دستور دوم، مجموعه‌ای به نام $s_1$ با اعضای ۱ تا ۶ ایجاد می‌نماید، دسـتور سـوم، نـوع $s_1$ (یعنـی، <class 'set'> ) را نمـایش مـی‌دهـد و دسـتور چهـارم، اعضای مجموعه $s_1$ (یعنی، مقادیر {1, 2, 3, 4, 5, 6} ) را نشان می‌دهد.

برای ایجاد مجموعه خالی، فقط می‌توانید از کلاس set (بدون پـارامتر) اسـتفاده کنیـد. بـه‌عنـوان‌مثـال، دستورات زیر را ببینید:

---

[1] Key [2] Key - Value [3] Unique

```
>>> s1 = set()
>>> type(s1)
>>> print(s1)
```

دستور اول، مجموعه تهی را با نام $s_1$ می‌کند، دستور دوم، نوع $s_1$ (یعنی، <class 'set'> ) را نمایش می‌دهد و دستور سوم، محتوی مجموعه $s_1$ (مقدار set() ) را نمایش می‌دهد. دقت کنید که برای ایجاد مجموعه خالی نمی‌توانید از عملگرهای { } استفاده کنید. زیرا، عملگرهای { }، برای ایجاد دیکشنری خالی به کار می‌روند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> d1 = {}
>>> type(d1)
>>> print(d1)
```

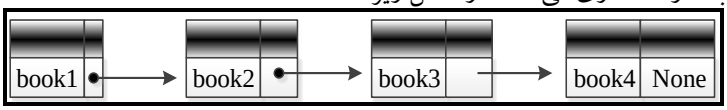
دستور اول، متغیری به نام $d_1$ با نوع دیکشنری ایجاد می‌کند، دستور دوم، نوع $d_1$ (یعنی، <class 'dict'>) را نشان می‌دهد که مشخص می‌کند یک دیکشنری است و دستور سوم، محتوی $d_1$ را نمایش می‌دهد که خالی است.

مجموعه‌ها از انوع نامنظم[1] و تغییرپذیر[2] پایتون هستند.

## ۲-۱-۱. ساختمان داده‌های پویا

ساختمان داده‌هایی هستند که تعداد عناصر آن‌ها در هنگام ترجمه مشخص نیست. به‌عنوان‌مثال، اگر بخواهید ساختمان داده‌ای داشته باشید که تعدادی شابک، نام کتاب و قیمت را نگه‌داری کند، باید از ساختمان‌های داده پویا استفاده کنید. چون، تعداد کتاب‌ها در زمان ترجمه مشخص نیست. انواع ساختمان داده‌های پویا عبارت انداز:

✪ **ساختمان داده‌های پویای ترتیبی (خطی)**، در این نوع ساختمان داده‌ها هر عنصر به عنصر قبلی یا بعدی‌اش اشاره می‌کند. نمونه‌ای از این ساختمان داده می‌توان **لیست پیوندی** را نام برد. لیست پیوندی انواع مختلف دارد که در فصل ۴ به آن‌ها می‌پردازیم. ساده‌ترین لیست پیوندی، لیست تک پیوندی است. در این نوع لیست پیوندی هر عنصر از دو بخش **داده**[3] و **پیوند**[4] تشکیل می‌شود. پیوند به داده بعدی لیست اشاره می‌کند. اما، مقدار پیوند آخرین عنصر لیست None است. نمونه‌ای از لیست پیوندی که اطلاعات کتاب‌ها را نگه‌داری می‌کند، در شکل زیر آمده است:




تعریف این ساختار به‌صورت زیر است:

```
class    ListBook :
    ISBN = “”
    bookName=””
    price = 0
    Link =None
```

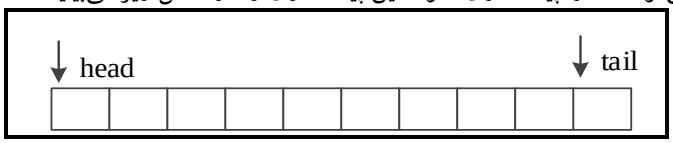
[1] Unordered [2] Mutable [3] Data [4] Link

✪ **ساختمان داده‌های غیر ترتیبی پویا (غیرخطی)**، در این ساختمان داده، عناصر به عنصر بعدی یا قبلی‌شان اشاره نمی‌کنند. به‌عنوان نمونه‌هایی از این نوع ساختمان داده می‌توان **درخت و گراف** را نام برد. در فصل‌های ۵ و ۶ با مفاهیم و پیاده‌سازی درخت و گراف آشنا خواهیم شد.

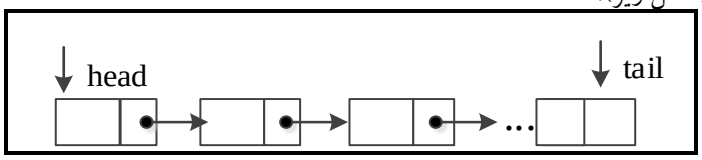
## ۳-۱-۱. ساختمان داده‌های نیمه ایستا

ساختمان داده‌های نیمه ایستا، ساختمان داده‌هایی هستند که گاهی اوقات از طریق ساختمان داده‌های ایستا و گاهی از طریق ساختمان داده‌های پویا قابل پیاده‌سازی هستند. در این ساختمان داده‌ها، اگر تعداد عناصر در هنگام ترجمه مشخص باشد، از طریق ساختمان داده‌های ایستا مانند آرایه می‌توان آن‌ها را پیاده‌سازی نمود. اما، اگر تعداد عناصر آن‌ها در زمان ترجمه مشخص نباشد، باید آن‌ها را از طریق ساختمان داده‌های پویا نظیر لیست پیوندی پیاده‌سازی نمود. این نوع ساختمان داده‌ها در زیر آمده‌اند:

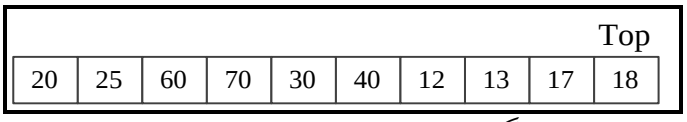
✪ **صف**[1]، ساختمان داده‌ای که در آن اولین ورودی اولین خروجی[2] باشد. یعنی، عناصر در **سر**[3] صف اضافه می‌شوند و از **انتهای**[4] آن حذف می‌شوند. اگر در هنگام ترجمه تعداد عناصر صف مشخص باشد، از طریق آرایه می‌توان آن را پیاده‌سازی نمود. این پیاده‌سازی را در شکل زیر می‌بینید:




اما، اگر در هنگام ترجمه تعداد عناصر صف مشخص نباشد، از طریق لیست پیوندی می‌توان آن را پیاده‌سازی نمود (شکل زیر):



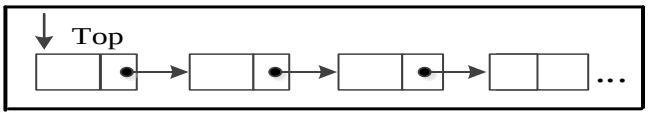

✪ **پشته**[5]، ساختمان داده‌ای که اولین ورودی آن، آخرین خروجی باشد[6] یا آخرین ورودی آن، اولین خروجی است[7]. اعضای پشته ازیک‌طرف آن اضافه یا حذف می‌شوند. پشته دارای اشاره‌گری به نام top است که به بالای پشته اشاره می‌کند (مکانی که عناصر پشته می‌توانند از آن مکان اضافه یا حذف شوند). اگر تعداد عناصر پشته در هنگام ترجمه مشخص باشد، پشته از طریق آرایه پیاده‌سازی می‌شود (شکل زیر):




اما، اگر تعداد عناصر پشته در هنگام ترجمه مشخص نباشد، باید از طریق لیست پیوندی پیاده‌سازی شود (شکل زیر):

---

[1].Queue [2]. First Input First Output (FIFO) [3]. Front [4].Rear
[5].Stack [6]. First Input Last Output(FILO) [7]. First Output Last Input (FOLI)




**ساختمان داده‌های ایستا و نیمه ایستا از نوع ساختمان داده‌های ترتیبی هستند.**

## ۲-۱. الگوریتم

الگوریتم، مجموعه‌ای از دستورالعمل‌ها که مراحل انجام کاری را به **زبان دقیق**، با **جزئیات کافی** بیان نماید، **ترتیب اجرای دستورات** و **شرط خاتمه** آن مشخص باشد.

همان‌طور که در این تعریف می‌بینید، برخی از ویژگی‌های الگوریتم عبارت‌اند از:

۱. **قطعیت**[1]**(عدم ابهام)**، دستورالعمل‌ها هیچ‌گونه ابهامی نداشته باشند. دستورالعمل‌ها به زبان دقیق بیان شوند.

۲. **پایان‌پذیر باشد**[2]، الگوریتم باید پس از انجام مراحل خاصی خاتمه یابد.

۳. **ترتیب اجرای آن مشخص باشد**، مراحل انجام دستورالعمل‌ها در الگوریتم باید مشخص باشد.

۴. **ورودی**[3]، الگوریتم می‌تواند ورودی داشته باشد یا ورودی نداشته باشد.

۵. **خروجی**[4]، الگوریتم باید حداقل یک خروجی داشته باشد.

۶. **کارایی**، الگوریتم باید قابل پیاده‌سازی باشد. یعنی، بتوان دستورالعمل‌های آن را به‌صورت دستی و گام‌به‌گام دنبال کرد.

الگوریتم‌ها توسط زبان‌های برنامه‌سازی پیاده‌سازی می‌شوند. با پیاده‌سازی الگوریتم توسط زبان‌های برنامه‌سازی، **برنامه**[5] ایجاد می‌شود. تفاوت بین برنامه و الگوریتم در زیر آمده است:

۱. الگوریتم باید خاتمه پذیر باشد. اما، برنامه می‌تواند خاتمه نداشته باشد (مانند سیستم‌عامل‌ها که هیچ‌گاه خاتمه نمی‌یابند).

۲. برنامه‌ها توسط رایانه‌ها قابل‌فهم و اجرا می‌باشند. اما، الگوریتم‌ها توسط رایانه‌ها قابل‌اجرا نیستند.

### ۱-۲-۱. کارایی الگوریتم

برای حل یک مسئله ممکن است الگوریتم‌های مختلفی وجود داشته باشد. به‌عنوان‌مثال، فرض کنید بخواهید الگوریتمی بنویسید که یک عدد را از ورودی خوانده، تعیین کند اول است یا خیر؟ برای تعیین عدد اول تعاریف زیر وجود دارد:

- عددی اول است که مجموع مقسوم‌علیه‌های آن برابر با یک باشد.
- عددی اول است که بر هیچ عدد کوچک‌تر از خودش به‌جز یک بخش‌پذیر نباشد.
- عددی اول است که بر هیچ عدد کوچک‌تر یا مساوی نصف خودش به‌جز یک بخش‌پذیر نباشد.

---

[1].Definiteness [2].Finitness [3].Input [4].Output [5]. Program

✪ عددی اول است که بر هیچ عدد اول کوچک‌تر از خودش بخش‌پذیر نباشد.

یا فرض کنید بخواهید محتوی متغیرهای x و y را تعویض کنید که به ترتیب مقادیر ۷ و ۱۰ دارند. برای حل این مسئله روش‌های مختلفی وجود دارد که عبارت‌اند از:

✪ **استفاده از متغیر کمکی برای تعویض مقادیر x و y**، این عمل به‌صورت زیر انجام می‌شود:

```
t = x  ⇨  t = 7
x = y ⇨  x = 10
y = t  ⇨  y = 7
```

✪ **استفاده از عملگرهای + و – برای تعویض مقادیر x و y**، این روش به‌صورت زیر انجام می‌شود:

```
x = x + y  ⇨ x = 7 + 10 = 17
y = x - y  ⇨  y = 17 – 10 = 7
x = x - y  ⇨ x = 17 – 7 = 10
```

✪ **استفاده از عملگرهای * و تقسیم برای تعویض مقادیر x و y**، این روش به‌صورت زیر عمل می‌شود:

```
x = x * y;  ⇨  x = 7 * 10 = 70
y = x // y;  ⇨  y = 70  // 10 = 7
x = x // y;  ⇨  x = 70 // 7 = 10
```

✪ **استفاده از انتساب چندگانه برای تعویض مقادیر x و y**، این روش به‌صورت زیر عمل می‌شود:

```
x , y = y, x
```

همان‌طور که در روش‌های مختلف حل مسائل بیان‌شده مشاهده کردید، کارایی الگوریتم به دو عامل اصلی بستگی دارد. ۱.**پیچیدگی حافظه**، مقدار حافظه‌ای که برای اجرای الگوریتم موردنیاز است. به‌عنوان‌مثال، در روش اول برای تعویض مقدار دو متغیر x و y به یک متغیر کمکی مانند t نیاز است. پیچیدگی حافظه یک الگوریتم با حرف S نمایش داده می‌شود. ۲. **پیچیدگی زمانی**، مدت‌زمانی که برای اجرای الگوریتم موردنیاز است. به‌عنوان‌مثال، هر یک از روش‌های تعیین عدد اول، دارای پیچیدگی زمانی متفاوتی هستند. پیچیدگی زمانی الگوریتم با حرف T نمایش داده می‌شود.

الگوریتمی کاراتر است که فضای حافظه کم‌تری را اشغال کند و جهت اجرا به زمان کم‌تری از پردازش‌گر نیاز داشته باشد. پیچیدگی الگوریتم را به دو روش می‌توان محاسبه کرد که عبارت‌اند از:

۱. **محاسبه دقیق زمان اجرا و حافظه مصرفی الگوریتم**، این روش نه‌تنها مشکل است، بلکه به نوع ماشینی که الگوریتم بر روی آن اجرا می‌شود، بستگی دارد. به‌عنوان‌مثال، یک داده با نوع صحیح در رایانه‌های ۱۶ بیتی، دو بایت از حافظه را اشغال می‌کند، ولی همین داده در رایانه‌های ۳۲ بیتی، چهار بایت از حافظه را اشغال می‌نماید. از طرف دیگر، همیشه به محاسبه دقیق زمان اجرا و حافظه مصرفی الگوریتم نیازی نیست.

۲. **محاسبه پیچیدگی الگوریتم به‌صورت تابعی از ورودی**، این روش نه‌تنها به نوع ماشین بستگی ندارد، بلکه بدون این‌که زمان اجرا و حافظه مصرفی الگوریتم به‌طور دقیق محاسبه شود، می‌تواند برای سنجش کارایی الگوریتم‌های مختلف به کار رود.

دستورالعمل‌های الگوریتم به دستورات برنامه تبدیل می‌شوند تا بتوانند توسط رایانه قابل‌اجرا باشند. دستورات برنامه دو نوع‌اند که عبارت‌اند از:

**۳. دستورات غیر اجرائی،** دستوراتی هستند که زمان پردازش‌گر را مصرف نمی‌کنند. این دستورات دارای گام صفر هستند. نمونه‌ای از این دستورات عبارت‌اند از:

**۱. دستورات تعریف تابع ۲. توضیحات برنامه ۳. دستورات تعریف کلاس‌ها.**

**مثال۱ – ۱. دستورات زیر در پایتون دارای گام صفر هستند.**

```
# Declare x, i;
def f1(x,  y):
class Square:
    …
```

**دستورات اجرایی،** هر دستور اجرای به ازای هر بار اجرا به یک گام نیاز دارد. یعنی، اگر دستور اجرای در یک حلقه تکرار قرار گیرد که n بار تکرار می‌شود، n گام را می‌گیرد.

**مثال۲– ۱. گام‌های اجرای دستورات زیر را به دست آورید:**

```
1)  sum = 0
2)  x = y * 3 + 5
3)  for i in range(1, n +1):
        x = x + 1
```

دستور اول، دارای ۱ گام است. چون، مقدار صفر را در متغیر sum قرار می‌دهد. دستور دوم، نیز گام یک را دارد. چون، یک دستور اجرای است. اما، گام دستور سوم به‌صورت زیر حساب می‌شود:

**i = 1⇨ گام 1**

**i+=1 ⇨ گام n**

پس for i in range(1, n+1) دارای n گام است. از طرف دیگر، x = x + 1 نیز n بار اجرا می‌شود، پس دارای n گام است. بنابراین، این حلقه تکرار دارای (n + n) = 2n گام است.

در دستورات شرطی(if)، عبارت شرطی دارای گام یک است. اما با توجه به نتیجه شرط (درست یا نادرست بودن) یکی از بخش‌های if یا else اجرا می‌شود و با توجه به تعداد دستورات if یا else گام‌های دستور محاسبه می‌گردد.

**مثال۳– ۱. گام‌های اجرای دستورات زیر را به دست آورید:**

```
x = 5  ⇨(1)
if  x < 10 :    ⇨ (1)
    x += 1      ⇨(1)
    y = x       ⇨(1)
  x -= 1        ⇨(1)
else:
    x = x – 1
```

گام این دستورات ۵ است. زیرا، دستور اول و دوم هرکدام دارای گام ۱ هستند. چون x کوچک‌تر از ۱۰ است، پس دستورات x += 1، y = x و x -= 1 اجرا می‌شوند که هرکدام دارای گام ۱ هستند.

**مثال۴– ۱. گام‌های اجرای حلقه‌های تودرتو زیر را به دست آورید:**

```
for i in range(1, n + 1):
    for j in range(1, m + 1):
```

```
    x = y
y -= 1
```

برای محاسبه گام این دستورات به‌صورت زیر عمل می‌شود:

- گام اجرای حلقه خارجی (n) است.
- گام اجرای حلقه داخلی (m) است. چون این حلقه n بار اجرا می‌گردد، پس n * m گام اجرای این حلقه است.
- دستور x = y دارای گام اجرای n * m است. بنابراین، گام اجرای این حلقه برابر است با :

n + nm + nm = 2nm + n

**مثال ۵ – ۱. گام اجرای تابع زیر را به دست آورید:**

```
def calculateSum (n):
    sum = 0
    for i in range(1, n + 1):
        sum += i
    return sum
```

گام این تابع به‌صورت زیر محاسبه می‌شود:

- دستور اول، دارای گام صفر است. چون تعاریف توابع گام صفر دارند.
- دستور دوم، sum = 0 دارای گام یک است.
- حلقه تکرار دارای گام (n) است.
- دستور sum += i دارای گام n است. چون n بار اجرا می‌شود.
- بنابراین، گام کل دستورات برابر 2n (n + n) است.

**مثال ۶ – ۱. گام اجرای دستورات زیر را به دست آورید:**

```
sum = 0
for i in range(n, 0, -1):
    sum += i
```

دستور sum = 0، دارای گام یک است، حلقه تکرار دارای گام n می‌باشد و دستور sum +=i، دارای گام n است. بنابراین، گام کل دستورات برابر با 2n = (n + n) است.

**مثال ۷ – ۱. گام اجرای قطعه برنامه زیر چند است:**

```
sum = 0
for i in range(0, n, 2):
    for j in range(m, 0, -3):
        sum += 1
```

دستور sum = 0، دارای گام یک است، دستور for با شمارنده i دارای گام $\frac{n}{2}$ می‌باشد، دستور for با شمارنده j دارای گام $\left(\frac{n}{2}\right)\left(\frac{m}{3}\right)$ است و دستور sum+=i دارای گام $\left(\frac{n}{2}\right) * \left(\frac{m}{3}\right)$ است. پس گام کل دستورات برابر با عبارت $T(n) = 1 + \left(\frac{n}{2}\right) + \left(\left(\frac{n}{2}\right)\left(\frac{m}{3}\right)\right) + \left(\frac{n}{2}\right)\left(\frac{m}{3}\right)$ است.

**مثال ۸ – ۱. گام اجرای دستورات زیر را به دست آورید:**

```
x = 0
for i in range(2, n+1, 5):
    for j in range(n, 2, -3):
        k = 1
        while( k <= n + 1) :
            x += 1
```

k *= 3

دستور x = 0، دارای گام یک است، حلقه for با شمارنده i دارای گام n + 5 می‌باشد، دستور for با شمارنده j، دارای گام $\left(\frac{n-3}{2}\right)$ (خود حلقه) ضرب در n + 4 (حلقه بالائی) است، حلقه while با شمارنده k دارای گام $log_3^{2n+1}$ (خود حلقه) * $\frac{n-3}{2}$ (برای حلقه با شمارنده j) *4 + n (برای حلقه با شمارنده i) است و دستور x+ =1 دارای گام $(log_3^{2n+1})$*$\frac{n-3}{2}$ * $(n+4)$ است. پس گام کل دستورات برابر است با:

$$T(n) = 1 + (n+5) + (n+4)(\left(\frac{n-3}{2}\right) + \frac{n-3}{2}\left((log_3^{2n+1}) + log_3^{2n+1}\right))$$

همان‌طور که مشاهده کرده‌اید، محاسبه گام اجرای دستورات کار سختی است. بنابراین، بهتر است از مرتبه اجرائی الگوریتم استفاده شود. مرتبه اجرائی را در ادامه می‌آموزیم.

## ۲-۲-۱. مرتبه اجرائی

برای مقایسه الگوریتم‌ها، سه نماد O و Ω و θ وجود دارد که در ادامه هرکدام از آن‌ها را توضیح خواهیم داد. از میان این نمادها به طول معمول از نماد O استفاده می‌شود. تعریف نماد O به‌صورت زیر است:

$$f(n) \in O(g(n)) \Leftrightarrow \exists n_0 > 0, c > 0 \quad \forall n \geq n_0 : f(n) \leq cg(n)$$

در این رابطه $n_0$ و c مقادیر ثابت هستند. $\exists c$، یعنی، حداقل یک c وجود دارد و $\forall n$ به معنای همه مقادیر n است. اگر $f(n) \in O(g(n))$، آن گاه g(n) یک کران بالا برای f(n) خواهد بود.

| **قضیه:** | **اگر $f(n) = a_m n^m + \cdots + a_1 n + a_0$ باشد، آنگاه $f(n) \in O(n^m)$ خواهد بود.** |
|---|---|

به‌عنوان‌مثال، تابع f(n) مقابل را در نظر بگیرید: f(n) = 4n + 4

$f(n) \in O(n)$ است. زیرا: $\forall n \geq 3 : 4n + 4 \leq 6n$

پس $n_0 = 3$ و $c = 5$ و $g(n) = n^2$ می‌باشد. بنابراین، می‌توان نتیجه گرفت که جمله غالب f(n) (جمله‌ای که رشد آن از دیگر جملات بیش‌تر باشد) برابر با $g(n)$ است. یعنی داریم:

$f(n) = 4n + 4 \in O(n)$
$f(n) = 10n^3 + 5n^2 + 6n + 4 \in O(n^3)$
$f(n) = n + log_2^n \in O(n)$
$f(n) = log_2^n + 5 \in O(log_2^n)$
$f(n) = 5 \in O(1)$

به‌عنوان‌مثال، تابع f(n) زیر را ببینید:

$f(n) = 5n^2 + 2n + 3$

$f(n) \in O(n^2)$ است. زیرا: $\forall n \geq 3 : 5n^2 + 2n + 3 \leq 5n^2$

پس $n_0 = 3$، $c = 6$ و $g(n) = n$ است.

**مثال ۹-۱. مرتبه اجرائی تابع زیر چیست:**

$$f(n) = \frac{n}{3}\left(\frac{n}{2} - 1\right) => \frac{n^2}{6} - \frac{n}{3} \in O(n^2)$$

مرتبه اجرائی برخی از توابع مهم به ترتیب از چپ به راست بر اساس میزان زمانی که مصرف می‌کنند، در جدول ۱-۱ آمده‌اند.

**اگر $f(n) \in O(g(n))$ باشد، آن‌گاه مرتبه اجرائی f(n)، برابر تمام توابع بزرگ‌تر از $g(n)$ نیز خواهد**

**بود. به‌عنوان‌مثال، مرتبه اجرائی توابع زیر دارای این شرایط هستند:**

$$f(n) = 4n^3 + n^2 + 6 \Rightarrow f(n) \in O(n^3) \Rightarrow f(n) \in O(n^4) => f(n) \in O(n^5)$$
...

**جدول ۱-۱ مرتبه اجرائی برخی از توابع مهم.**

| توانی | فاکتوریل | نمایی | چندجمله‌ای | - | خطی | لگاریتمی | ثابت | تابع |
|---|---|---|---|---|---|---|---|---|
| $O(n^n)$ | O(n!) | $O(a^n)$ | $O(n^a)$ | O(nlogn) | O(n) | O(logn) | O(1) | مرتبه اجرائی |

برای محاسبه مرتبه اجرائی (O) به نکات زیر توجه کنید:

✪ دستورات انتساب، محاسباتی، رابطه‌ای، منطقی شرطی (نظیر if) و دستورات ورودی و خروجی **پیچیدگی زمانی ثابت** دارند.

✪ دستورات تکرار مانند for، while و do while درصورتی‌که تعداد تکرار مستقل از اندازه ورودی باشد، پیچیدگی زمانی ثابتی دارند، در غیر این صورت، پیچیدگی زمانی آن‌ها به تعداد تکرار بستگی دارد.

✪ دستورات تکرار متداخل پیچیدگی زمانی آن‌ها در هم ضرب می‌شود.

✪ تعاریف توابع، کلاس‌ها و توضیحات پیچیدگی زمانی صفر دارند.

**مثال ۱۰-۱. مرتبه اجرائی دستورات زیر را به دست آورید:**

```
sum = 0
for i in range(1, n + 1):
    sum += i
```

مرتبه اجرائی این دستورات به‌صورت زیر است:

چون، دستورات sum = 0 و sum += i دارای O(1) هستند و حلقه‌for، دارای O(n) می‌باشد. بنابراین، O(n) + O(1) = O(n) است.

**مثال ۱۱-۱. مرتبه اجرائی دستورات زیر را به دست آورید:**

```
sum = 0
for i in range(1, n + 1):
    k = 0
    for j in range (i, n):
        k = k + 1
```

دستورات sum = 0، k =0 و k = k + 1 دارای O(1) هستند. چون دو حلقه (متداخل)داریم، پس $O(n^2)$ را برای حلقه‌ها خواهیم داشت. بنابراین، O(1) + $O(n^2)$ برابر با $O(n^2)$ خواهد بود.

**مثال ۱۲-۱. پیچیدگی زمانی دستورات زیر را به دست آورید:**

```
sum = 0
while n > 0 :
    sum += n % 10
    n //= 10
```

پیچیدگی زمانی این الگوریتم به تعداد تکرار while بستگی دارد. همان‌طور که می‌بینید، n در داخل حلقه تکرار تقسیم بر ۱۰ می‌گردد. پس اگر while، k بار اجرا شود، آنگاه $10^k$ <= n خواهد شد. بنابراین، تعداد تکرار این الگوریتم برابر O(k) است:

$$n \geq 10^k \Rightarrow k = log_{10}^n \Rightarrow O(log_{10}^n)$$

**مثال ۱۳ – ۱. پیچیدگی الگوریتم زیر را تعیین کنید:**

```
x = 0
i = n
while (i > 1) :
   sum += i % 10
    i = i % 2
```

پیچیدگی دستورات x = 0 و i = n برابر O(1) است. اما، پیچیدگی دستورات داخل حلقه تکرار به تعداد تکرار آن‌ها بستگی دارد. چون شرط حلقه i > 1 است. از طرف دیگر، دستورات i %= 2 نتیجه آن 0 یا 1 است. پس حلقه تکرار حداکثر یک‌بار اجرا خواهد شد و پیچیدگی زمانی این دستورات نیز یک است. بنابراین، پیچیدگی زمانی کل دستورات O(1) می‌باشد.

## نماد Ω

نماد Ω عبارت است از: $\mathbf{f(n) \geq cg(n)} \quad \mathbf{f(n) \in \Omega(g(n)) \leftrightarrow \exists c, n_0 > 0, \forall_n > n_0}$.

اگر $f(n) \in \Omega(g(n))$ باشد،آن گاه g(n) یک کران پایین برای f(n) خواهد بود.

**قضیه:** اگر $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$،آن گاه $f(n) \in \Omega(n^m)$ خواهد بود.

به‌عنوان‌مثال، در تابع $f(n) = 6n^3 + 2n^2 + 5$ داریم: $f(n) \in \Omega(n^3)$. چون، $\forall_n \geq 1: f(n) \geq 6n^3$. پس، $g(n) = n^3, c = 6, n_0 = 1$ را خواهیم داشت.

یا اگر $n^2 + n \log n$ باشد، آن گاه $f(n) \in \Omega(n^2)$ را داریم. چون، $\forall_n > 1: n^2 +$ است $n \log n \geq n^2$. پس، $g(n) = n^2, c = 1, n_0 = 1$ را خواهیم داشت . مثال‌های زیر را ببینید:

$f(n) = 10n^3 + 5n^2 + 6n \in \Omega(n^3)$, $f(n) = n + \log_2^n \in \Omega(n)$,
$f(n) = \log_2^n + 5 \in \Omega(\log_2^n)$, $f(n) = 5 \in O(1)$

**نکته:** اگر $f(n) \in \Omega(g(n))$ باشد،آن گاه مرتبه اجرایی f(n) برابر تمام توابع کوچک‌تر از g(n) نیز خواهد بود(مانند):

$f(n) = 14n^6 + 5n^4 + 6n^3 => f(n) \in \Omega(n^6) => f(n) \in \Omega(n^5) => f(n) \in \Omega(n^4) => \cdots$

## نماد θ

نماد Θ عبارت است از: $f(n) \in \Theta(g(n)) <=> \exists c_1, c_2 n_0 > 0, \forall_n c_1 g(n) \leq f(n) \leq c_2 g(n)$.

نماد Θ تابع f(n) را از بالا و پایین محدود می‌نماید. درجه رشد تابع f(n) و g(n) با هم برابر است.

قضیه: اگر $f(n) = a_m n^m + a_{m-1} n^{m-1} + \cdots + a_1 n + a_0$ باشد، آن گاه $f(n) \in \Theta(n^m)$ خواهد بود. به‌عنوان‌مثال، تابع $f(n) = 4n^2 + 1$ باشد، آن گاه $f(n) \in \Theta(n^2)$ خواهد بود.

چون $\forall_n > 1: 4n^2 \leq f(n) \leq 5n^2$ است. پس، $g(n) = n^2, c_2 = 5, c_1 = 4$ است.

یا تابع $f(n) = 7n^5 + n^2$ باشد، آن گاه $f(n) \in \Theta(n^5)$ خواهد بود. چون $\forall_n >: 7n^5 \leq f(n) \leq 8n^5$ است. پس، $g(n) = n^5, c_2 = 8, c_1 = 7, n_0 = 0$ است . مثال‌های زیر را ببینید:

$f(n) = 10n^3 + 5n^2 + 6n \in \Theta(n^3)$, $f(n) = n\log_2^n \in \Theta(n)$,
$f(n) = \log_2^n + 5 \in \Theta(\log_2^n)$, $f(n) = 5 \in \Theta(1)$

**نکته:** $f(n) \in \Omega(g(n))$، $f(n) \in O(g(n))$ است، اگر و تنها اگر $f(n) \in \Theta(g(n))$ باشد.

**نکته:** $f(n) \in \Theta(g(n))$ است، اگر و تنها اگر $g(n) \in \Theta(f(n))$ باشد.

**نکته:** $f(n) \in O(g(n))$ است، اگر و تنها اگر $g(n) \in \Omega(f(n))$ باشد.

## ۳-۱. توابع بازگشتی

توابع بازگشتی، توابعی هستند که خودشان را به‌طور مستقیم یا غیرمستقیم فراخوانی می‌کنند. معمولاً مسائل تکراری را به دو روش می‌توان حل نمود:

۱. استفاده از دستورات تکرار for، while، do while.

۲. استفاده از روش بازگشتی

معمولاً کارایی روش بازگشتی ازلحاظ زمان اجرا از روش تکرار کم‌تر است. زیرا، در روش بازگشتی با هر فراخوانی تابع، سربار اضافی ایجاد خواهد شد (آدرس بازگشت و پارامترها در پشته نگه‌داری می‌شوند). باوجوداین، در بسیاری از موارد، روش‌های بازگشتی راه‌حل‌های ساده‌تر و راحت‌تری نسبت به راه‌حل‌های تکراری ارائه می‌کنند. به همین دلیل، روش بازگشتی ابزار مفیدی برای حل مسائل برنامه‌نویسی محسوب می‌شود. روش بازگشتی در حل مسائل غیر عددی نظیر نوشتن کامپایلرها، جست‌وجو و مرتب کردن در الگوریتم‌ها به کار می‌رود.

| | |
|---|---|
| **نکته:** | **همان‌طور که بیان گردید، الگوریتم‌های بازگشتی حافظه بیش‌تری مصرف می‌کنند و سرعت آن‌ها پایین‌تر است. اما، کد پیاده‌سازی آن‌ها کم‌تر است و راحت‌تر می‌توان آن‌ها را پیاده‌سازی نمود.** |

الگوریتم بازگشتی از دو بخش تشکیل‌شده است که عبارت‌اند از:

۱. **قانون بازگشتی**، فرمولی است که بر اساس آن الگوریتم بازگشتی خودش را فراخوانی می‌نماید.

۲. **شرط خاتمه**، شرطی که به فراخوانی متد بازگشتی خاتمه می‌دهد.

اکثر مسائل ریاضی را می‌توان به روش بازگشتی حل نمود. به‌عنوان‌مثال، روش بازگشتی n! به‌صورت زیر است:

$$n! = n * \underbrace{(n-1) * (n-2) * \ldots * 2 * 1}_{(n-1)!} = n * (n-1)!$$

$$(n-1)! = (n-1) * \underbrace{(n-2) * \ldots * 2 * 1}_{(n-2)!} = (n-1) * (n-2)!$$

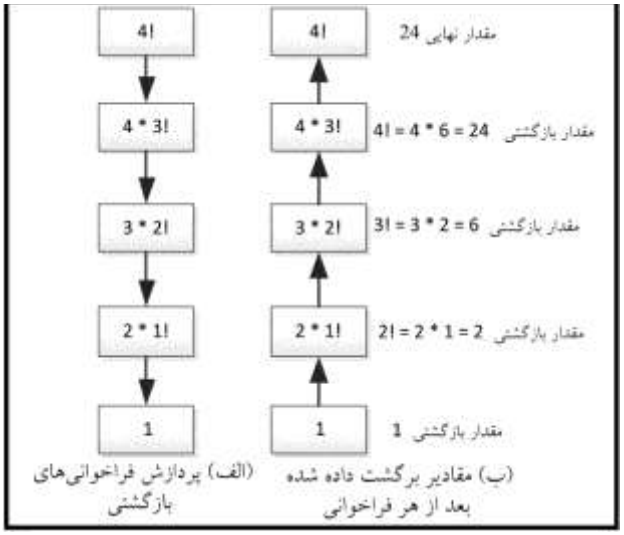
...

$$1! = 1 * 0!$$
$$0! = 1$$

اگر به این تعاریف دقت کنید، قانون بازگشتی n! برابر با $n * (n-1)!$ است و شرط خاتمه روش بازگشتی n!، اگر n برابر صفر شود، می‌باشد که یک را برمی‌گرداند. بنابراین، فرمول بازگشتی n! را می‌توان به‌صورت زیر نوشت:

$$n! = \begin{cases} 1 & \text{اگر } n == 0 \text{ باشد} \\ n * (n-1)! & \text{وگرنه} \end{cases}$$

روش اجرای !4 را در شکل ۲-۱ می‌بینید.




**شکل ۲-۱ روش محاسبه !4 را نمایش می‌دهد.**

به‌عنوان‌مثال دیگر، a و b دو عدد صحیح بزرگ‌تر از صفر باشند، فرمول محاسبه $a^b$ به‌صورت زیر می‌باشد:

$$a^b = \underbrace{a * a * a * \dots * a}_{b \text{ مرتبه}} = a * \underbrace{a * a * \dots a}_{b-1 \text{ مرتبه}} = a * a^{b-1}$$

$$a^{b-1} = \underbrace{a * a * a * \dots * a}_{b-1 \text{ مرتبه}} = a * \underbrace{a * a * \dots * a}_{b-2 \text{ مرتبه}} = a * a^{b-2}$$

$$\dots$$

$$a^1 = a * 1 = a * a^0$$

$$a^0 = 1$$

پس، داریم:

$$a^b = \begin{cases} 1 & \text{اگر } b == 0 \text{ باشد} \\ a * a^{b-1} & \text{وگرنه} \end{cases}$$

یا فرمول بازگشتی مجموع اعداد یک تا n به‌صورت زیر است:

$$Sum = n + \underbrace{(n-1) + (n-2) + \dots + 1}_{Sum(n-1)} = n + Sum(n-1)$$

$$Sum(n-1) = (n-1) + \underbrace{(n-2) + (n-3) + \dots + 1}_{Sum(n-2)} = (n-1) + Sum(n-2)$$

$$\dots$$

$$Sum(1) = 1 + Sum(0)$$

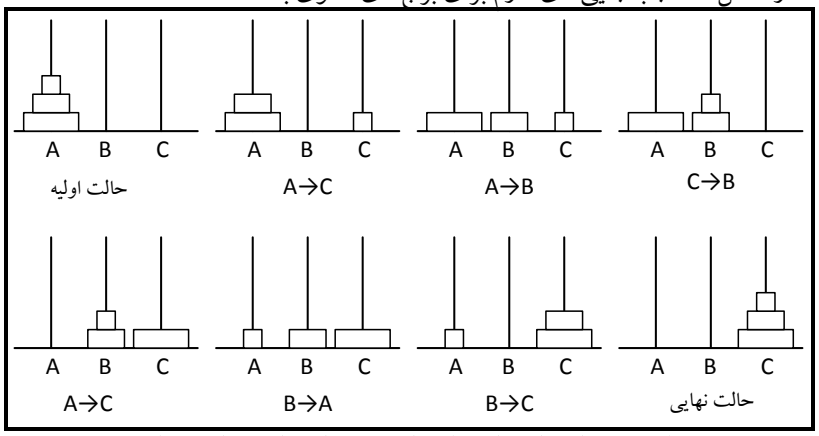
$$Sum(0) = 0$$

پس داریم:

$$Sum(n) = \begin{cases} 0 & \text{اگر } n == 0 \text{ باشد} \\ n + Sum(n-1) & \text{وگرنه} \end{cases}$$

**برج‌های هانوی**

یکی از مسائل معروف بازگشتی، برج‌های هانوی است. در برج‌های هانوی سه برج (میله) و n حلقه (دیسک) داریم. حلقه‌ها به ترتیب نزولی از پایین به بالای میله‌ی اول قرار دارند. هدف، انتقال حلقه‌ها از میله‌ی اول به میله‌ی سوم به کمک میله‌ی دوم است. در این جابه‌جایی‌ها باید دو شرط رعایت شود: ۱ - در هیچ مرحله‌ای حلقه‌ی کوچک‌تر، پایین‌تر از حلقه‌ی بزرگ‌تر قرار نگیرد. ۲- در هر مرحله، تنها یک حلقه جابه‌جا شود. در شکل۳-۱ جابه‌جایی‌های لازم برای برج‌های هانوی با سه حلقه نشان داده‌شده است.

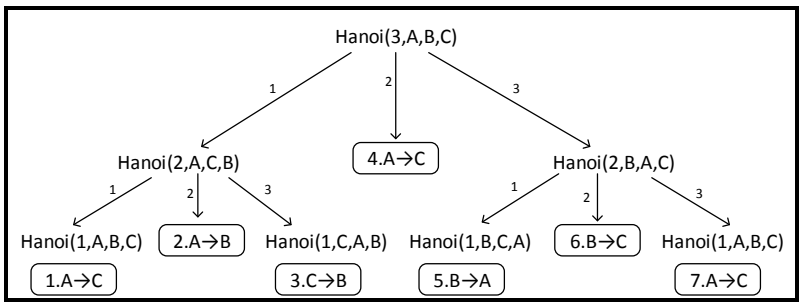



شکل۳-۱ جابه‌جایی‌های حلقه‌ها در برج‌های هانوی با سه حلقه.

برای حل مسئله برج‌های هانوی با n حلقه نیاز به $2^n - 1$ جابه‌جایی است. پس، پیچیدگی آن $O(2^n)$ است که برای nهای بزرگ بسیار وقت‌گیر خواهد بود. برج‌های هانوی را می‌توان با استفاده از الگوریتم زیر پیاده‌سازی نمود:

```
def    Hanoi(n, A, B, C):
   if (n==1):
     print("move a disk from" ,A ,"to",C ) //  انتقال یک حلقه از مبدأ به مقصد
   else:
      Hanoi(n-1, A, C, B)        //    انتقال n-1 حلقه از مبدأ به کمکی
      print( "move a disk from", A, "to", C)  //     انتقال یک حلقه از مبدأ به مقصد
     Hanoi(n-1,B,A,C);          //     انتقال n-1حلقه از کمکی به مقصد
```

در این الگوریتم n تعداد حلقه و میله‌ها بدون در نظر گرفتن نام آن‌ها، به ترتیب میله مبدأ، کمکی و مقصد هستند. همان‌طور که در الگوریتم نشان داده‌شده است، در هر مرحله به‌صورت بازگشتی سه عمل انجام می‌شود: ۱ . انتقال n-1 حلقه از مبدأ به کمکی ۲. انتقال حلقه باقی‌مانده از مبدأ به مقصد ۳. انتقال n-1 حلقه از کمکی به مقصد. در شکل۴-۱ مراحل اجرای برج‌های هانوی با سه حلقه نشان داده‌شده است.




شکل ۴-۱ مراحل اجرای برج‌های هانوی با سه حلقه.

### حل روابط بازگشتی با استفاده از روش تکرار با جای گذاری

یکی از روش‌های حل بعضی از روابط بازگشتی، روش تکرار با جای گذاری است. در این روش، با استفاده از رابطه‌ی بازگشتی و جای گذاری nهای مختلف، درنهایت جای گذاری مقدار ثابت به جواب نهایی خواهیم رسید.

**مثال ۱۴-۱. رابطه‌ی بازگشتی زیر را به روش تکرار با جایگذاری حل نمایید:**

$$\mathbf{T(n)} = \begin{cases} \mathbf{a} & \mathbf{n == 2} \\ \mathbf{T(n-b)} & \mathbf{n > 2} \end{cases}$$

$T(n) = T(n-b)+c$

$= (T(n-b-b)+c = T(n-2b)+2c$

…

$= T(n-ib)+ic$

به فرمولی برای جمله‌ی iام رسیدیم. حال آن را برابر مقدار آستانه قرار می‌دهیم تا i به دست آید:

$n - ib = 2 \rightarrow i = \frac{n-2}{b}$

اکنون i را در جمله‌ی عمومی قرار می‌دهیم تا به فرمول T(n) برسیم:

$T(n) = T(n - ib) + ic \quad \rightarrow \quad T(n) = T(2) + \frac{n-2}{b} = a + \frac{n-2}{b} \in O(n)$

**مثال ۱۵-۱. رابطه‌ی بازگشتی زیر را به روش تکرار با جایگذاری حل کنید:**

$$\mathbf{T(n)} = \begin{cases} \mathbf{a} & \mathbf{n = 1} \\ \mathbf{bT\left(\frac{n}{b}\right) + d} & \mathbf{n > 1} \end{cases}$$

$T(n) = bT(\frac{n}{b})+d$

$= b(bT(\frac{n}{b^2})+d = b^2\, T(\frac{n}{b^2})+(b+1)d$

$= b^2(bT(\frac{n}{b^3})+d)+(b+1)d = b^3\, T(\frac{n}{b^3})+(b^2+b+1)d$

…

$= b^iT(\frac{n}{b^i})+(b^{i-1} + \cdots + b + 1)d = b^i\, T\left(\frac{n}{b^i}\right) + (b^i - 1)d$

$$\frac{n}{b^i} = 1 \rightarrow n = b^i \rightarrow i = \log_b^n$$

i را در جمله عمومی قرار می‌دهیم:

$T(n)=b^i\, T\left(\frac{n}{b^i}\right) + (b^i - 1)d$

$\Rightarrow T(n) = b^{\log_b^n}\, T(1) + \left(b^{\log_b^n} - 1\right)d$

$\Rightarrow T(n) = an + (n-1)d$

$\Rightarrow T(n) \in O(n)$

**مثال ۱۶-۱. رابطه بازگشتی زیر را به روش تکرار با جایگذاری حل نمایید:**

$$T(n) = \begin{cases} a & n = 2 \\ b \quad T\left(\frac{n}{b}\right) + nd & n > 1 \end{cases}$$

$$T(n) = b\,T\left(\frac{n}{b}\right) + nd$$

$$= b\left(b\,T\left(\frac{n}{b^2}\right) + \frac{n}{b}d\right) + nd = b^2\,T\left(\frac{n}{b^2}\right) + 2nd$$

$$= b^2\left(b\,T\left(\frac{n}{b^3}\right) + \frac{n}{b^2}d\right) + 2nd = b^3\,T\left(\frac{n}{b^2}\right) + 3nd$$

$$= b^i\,T\left(\frac{n}{b^i}\right) + ind$$

$$\rightarrow \frac{n}{b^i} = 1 \rightarrow n = b^i \rightarrow i = \log_b^n$$

$$\rightarrow \mathbf{T(n) = b^i T\left(\frac{n}{b^i}\right) + ind \rightarrow T(n) = n\,T(1) + (\log_b^n)nd}$$

$$\mathbf{=> T(n) = an + dn(\log_b^n) \rightarrow T(n) \epsilon O(n \log n)}$$

**مثال ۱۷–۱. تابع زمانی و پیچیدگی زمانی تابع زیر را محاسبه نمایید.**

```
def   func(n):
  if (n == 1):
     return 1
  else:
    return  n + func(n - 1)
```

رابطه بازگشتی این تابع به‌صورت زیر است:

$$\mathbf{T(n)} = \begin{cases} \mathbf{1} & \mathbf{n = 1} \\ \mathbf{n + T(n-1)} & \mathbf{n > 1} \end{cases}$$

$$\mathbf{T(n) = T(n-1) + n}$$

$$\mathbf{= T\big((n-1) + n - 1\big) + n = T(n-1) + n + (n-1)}$$

$$\mathbf{= T\big((n-2) + n - 2\big) + n + (n-1) = T(n-1) + n + (n-1) + (n-2)}$$

$$\mathbf{\cdots}$$

$$\mathbf{T\big(\big(n - (n-1)\big)\big) + n + (n-1) + \cdots + (n - (n-1))}$$

$$\mathbf{T(1) + n + n + (n-1) + \cdots + 1}$$

$$\mathbf{T(n) = \frac{n(n+1)}{2} + 1 \rightarrow T(n)\ \epsilon\ O(n^2)}$$

## ۴–۱. مسائل حل‌شده

**مثال ۱. تابع جست‌وجوی یک مقدار در یک آرایه نامرتب به طول n عنصر در زیر آمده است. پیچیدگی زمانی این تابع را در بهترین حالت، بدترین حالت و حالت میانگین به دست آورید. میزان حافظه مصرفی را نیز به دست آورید.**

```
def IndexOf(a, value):
   n = len(a)
   for i in range(0, n):
      if (a[i] == value):
         return i
   return –1
```

بهترین حالت زمانی اتفاق می‌افتد که اولین عنصر آرایه با مقدار value برابر باشد. پس، پیچیدگی زمانی آن O(1) است. چون فقط یک مقایسه انجام می‌شود. اما، بدترین حالت زمانی رخ می‌دهد که مقدار value آخرین عنصر آرایه باشد یا مقدار value در آرایه موجود نباشد. در این حالت n مقایسه انجام خواهد شد. پس، پیچیدگی زمانی آن O(n) است. ولی، اگر value در وسط آرایه باشد، حالت میانگین اتفاق می‌افتد. در این حالت n / 2 مقایسه انجام می‌شود. پس، پیچیدگی زمانی آن O(n / 2) خواهد شد. پیچیدگی زمانی O(n / 2) همان پیچیدگی O(n) است.

پیچیدگی حافظه به‌صورت زیر محاسبه می‌شود:

**۱.** متغیرهای n، value و i قسمت ثابت حافظه مصرفی هستند که دارای پیچیدگی S(1) = O(1) هستند.

**۲.** آرایه a قسمت متغیر حافظه مصرفی است که دارای طول n می‌باشد. پس، پیچیدگی حافظه آن S(n) = O(n) می‌باشد.

**مثال ۲. پیچیدگی زمانی و مقدار حافظه مصرفی تابع ()Reverse زیر که عدد n را به‌عنوان پارامتر دریافت می‌کند و مقلوب آن را برمی‌گرداند، به دست آورید:**

```
def Reverse(n):
    r = 0
    while  n > 0 :
        r = r * 10 + n % 10
        n //= 10
    return r
```

پیچیدگی زمانی دستورات r = 0، r = r * 10 + n % 10 و n //= 10، برابر O(1) است. پس پیچیدگی زمانی این تابع به تعداد تکرار دستورات while بستگی دارد. در داخل while، هر بار n بر ۱۰ تقسیم می‌گردد (پس در بدترین حالت، حلقه while به تعداد ارقام n تکرار می‌شود). پس در بدترین حالت $n >= 10^m$ است که m تعداد تکرار while است. بنابراین، پیچیدگی زمانی تابع برابر با $O(log_{10}^{n})$ است.

در این تابع متغیرهای ثابت n و r استفاده شده‌اند. پس پیچیدگی حافظه آن S(1) = O(1) می‌باشد.

**مثال ۳. پیچیدگی زمانی و میزان حافظه مصرفی تابع مرتب‌سازی ()bubbleSort زیر را به دست آورید:**

```
def bubbleSort(a, n):
    for i in range(n):
        for j in range( i + 1, n):
            if (a[i] < a[j]):
                a[i], a[j] = a[j], a[i]
```

پیچیدگی زمانی این تابع برابر با تعداد تکرار حلقه‌های for است. دستور for خارجی دقیقاً n – 1 بار تکرار می‌شود. حلقه for داخلی نیز در بدترین حالت n - 1 مرتبه تکرار خواهد شد. پس پیچیدگی زمانی آن برابر با:

$$T(n) = O((n-1)(n-1)) = O(n^2 - n - n + 1) = O(n^2) - O(2n) + O(1) = O(n^2)$$

میزان حافظه مصرفی این تابع از دو بخش زیر تشکیل‌شده است:

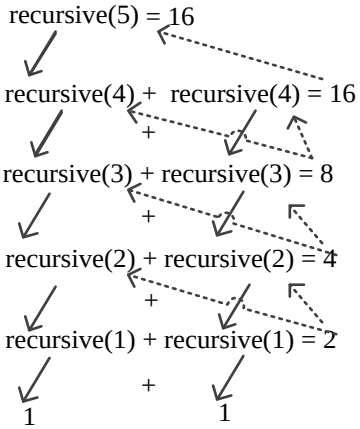
**۱. بخش ثابت حافظه مصرفی**، شامل متغیرهای n، i و j می‌باشد که پیچیدگی آن‌ها S(1) = O(1) است.

**۲. بخش متغیر حافظه مصرفی**، شامل آرایه a است که دارای n عنصر می‌باشد. پس، پیچیدگی حافظه آن S(n) = O(n) می‌باشد.

**مثال ۴. مقدار recursive(5) تابع بازگشتی زیر را به دست آورید:**

```
def recursive(n):
    if  n == 1 :
        return 1
    else:
        return (recursive(n – 1) + recursive(n – 1))
```

نمودار درختی اجرای این تابع به‌صورت زیر است:




**مثال ۵. فرض کنید n توانی از ۲ باشد، تابع زیر چه عددی را به‌عنوان جواب برمی‌گرداند:**

```
def   test(n):
    if  n == 1 :
       return  1
    return ( 2 * test(n / 2) + 6 * n – 1)
```

$$T(n) = 2T\left(n/2\right) + 6n - 1, \quad T(n) = 6n\log_2^n + 1$$

$$T(n) = 2\left[\left(6\frac{n}{2}\right)\log_2\frac{n}{2} + 1\right] + 6n - 1$$

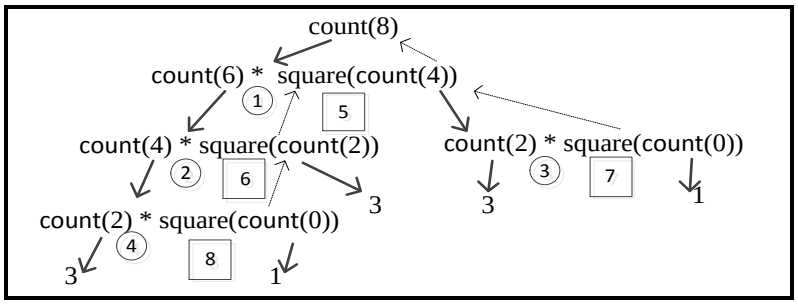
$$T(n) = 6n\left(\log_2^{n-1}\right) + 2 + 6n - 1$$

$$T(n) = 6n\log_2^n - 6n + 2 + 6n - 1 = 6n\log_2^n + 1$$

**مثال ۶. در فراخوانی تابع زیر برای n = 8 چند عمل ضرب انجام می‌شود؟ فرض کنید هر عمل square نیز به یک عمل ضرب نیاز دارد.**

```
def  count(n):
    if  n <= 0:
       return  1
    if  n == 1:
       return  2
    if  n == 2:
       return  3
    return (count(n – 2) * square(count(n – 4)))
```

درخت بازگشتی به‌صورت زیر است:

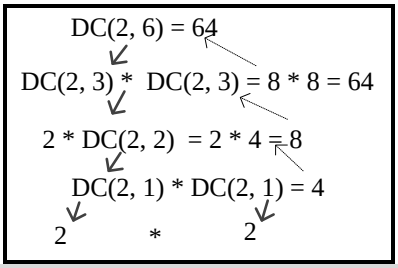


همان‌طور که در شکل می‌بینید، ۴ عمل ضرب انجام‌شده است(شماره‌های موجود در دایره) و چهار مرتبه square تکرار شده است(شماره‌های موجود در مربع). ازآنجایی‌که هر square یک عمل ضرب می‌باشد. بنابراین، ۸ عمل ضرب انجام‌شده است.

**مثال ۷. تابع زیر برای مقادیر a = 2 و n = 6 چه مقداری را برمی‌گرداند:**

```
def Dc(a, n):
   if n == 1:
      return a
   if n % 2 == 0 :
      return(DC(a, n // 2) * DC(a, n//2))
   return (a * DC(a, n – 1))
```

درخت بازگشتی تابع به‌صورت زیر است:




**مثال ۸. تابع زیر را در نظر بگیرید:**

```
def combination(n, m):
    if (m == n) or (m == 0):
      return 1
    else:
      return(combination(n –1, m) + combination(n–1, m-1))
```

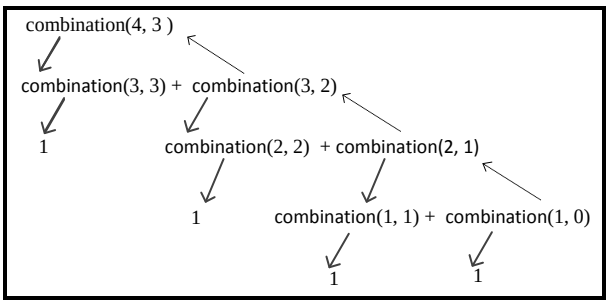
**برای m < n دلخواه عمل + چند بار اجرا می‌شود.**

فرمول بازگشتی این تابع به‌صورت زیر است:

$$\binom{n}{0} = \binom{n}{n} = 1 \qquad \text{اگر } m == n \text{ یا } m == 0 \text{ باشد،}$$

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1} \qquad \text{وگرنه}$$

بنابراین این تابع را به مسائل کوچک‌تری تقسیم می‌کنیم تا به حالتی برسیم که m == n شود یا m == 0 شود. در این حالت تابع ۱ را برمی‌گرداند، درنهایت یک‌های به‌دست‌آمده را جمع می‌کنیم. به‌عنوان‌مثال، اگر n=4 و m=3 باشد، درخت بازگشتی به‌صورت زیر به دست می‌آید:



همان‌طور که مشاهده می‌شود، مجموع یک‌ها برابر با ۴ است که برابر با $\binom{n}{m} - 1$ است.

**مثال ۹. تابع زیر را در نظر بگیرید. تعداد فراخوانی‌های بازگشتی چقدر است؟**

```
def C(n, k):
    if (k == 0) or (k == n):
        return 1
    else:
        return (C(n – 1, k-1)+ C(n–1, k))
```

این مثال مانند مثال ۸ است و به همان روش حل می‌گردد.

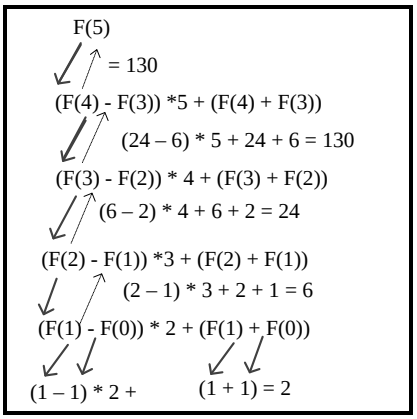
**مثال ۱۰. مقدار بازگشتی F(5) چیست؟**

```
def F(n):
    if n < 2:
        return 1
    else:
        return ((F(n – 1) - F(n - 2))*n +(F(n–1)+ F(n-2)))
```

فرمول تابع به‌صورت زیر می‌باشد:

$$F(n)\begin{cases}1 & \text{اگر } n < 2\\ ((F(n-1)-F(n-2))*n+(F(n-1)+F(n-2)) & \text{وگرنه}\end{cases}$$

اکنون برای *f(5)* درخت فراخوانی‌های بازگشتی را رسم می‌کنیم:

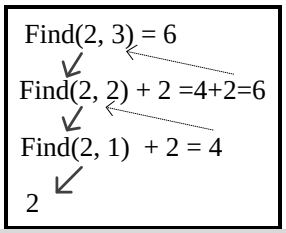



پس، مقدار بازگشتی ۱۳۰ است.

**مثال ۱۱. تابع زیر را در نظر بگیرید. اگر a و b اعداد صحیح باشند، تابع چه مقداری را برمی‌گرداند؟**

```
def Find(a, b):
    if b == 1:
        return a
    else:
        return (Find(a, b – 1) + a)
```

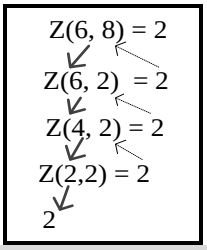
این تابع همان a * b (a را b بار با خودش جمع می‌کند) است. یعنی، اگر a = 2 و b = 3 باشد، درخت بازگشتی تابع به‌صورت زیر است:




**مثال ۱۲. تابع زیر با مقدار n = 8 و k = 6 چه مقداری را برمی‌گرداند؟**

```
def  Z(k,  n):
    if  n == k:
        return k
    elif  n> k:
        return (Z(k, n – k))
    else:
       return Z(k – n, n)
```

درخت بازگشتی تابع به‌صورت زیر است:




**مثال۱۳. در قطعه برنامه زیر مقدار محاسبه‌شده برای total را تعیین کنید:**

```
total = 0
for i in range(1, n + 1) :
  k = n
  while k != 1:
     k = k // 2
     total += 1
```

مقدار total برابر است با تعداد دفعاتی که دستور total += 1 اجرا می‌شود. دستور total += 1 در داخل حلقه while قرار دارد که این حلقه $\log_2^k$ (همان $\log_2^n$، چون k در ابتدای حلقه while برابر n است) مرتبه اجرا می‌شود. چون، این حلقه در داخل حلقه for قرار دارد و حلقه for، به تعداد n مرتبه اجرا می‌شود، پس دستور total += 1 به تعداد $n * \log_2^n$ مرتبه اجرا خواهد شد. از آنجائی که هر مرتبه به total یک واحد اضافه می‌شود، پس مقدار total برابر با $n * \log_2^n$ خواهد شد.

**مثال۱۴. در قطعه برنامه زیر t = t+1 چند بار اجرا می‌شود:**

```
for i  in range(1, m + 1):
    for j  in range(1, (m – i) + 1):
    t = t +1
```

تعداد اجرای $t = t + 1$ برابر با تعداد تکرار حلقه داخلی است و حلقه داخلی به‌صورت زیر تکرار می‌شود:

| i | 1 | 2 | … | m |
|---|---|---|---|---|
| تعداد تکرار حلقه داخلی | m – 1 | m – 2 | | 0 |

$$(\mathbf{m-1})+(\mathbf{m-2})+(\mathbf{m-3})+(\mathbf{m-4})+\ \ldots \mathbf{2}+\mathbf{1}=\frac{(\mathbf{m-1})\times(\mathbf{m-1+1})}{\mathbf{2}}=\frac{(\mathbf{m-1})\mathbf{m}}{\mathbf{2}}$$

**مثال۱۵. برای m > 0، پس از اجرای قطعه برنامه زیر، مقدار x چند خواهد شد؟**

```
X = 0
for  i  in range(m + 1):
    for j in range(i + 1):
        for k in range(1, m + 1):
            x = x + 1
```

مقدار x برابر با تعداد اجرای x = x + 1 است. ازآنجایی‌که x = x + 1 در حلقه داخلی قرار دارد، بنابراین تعداد تکرار آن برابر است با:

(تعدد تکرار حلقه با شمارنده k) * (تعداد تکرار حلقه با شمارنده j)

حلقه با شمارنده j، برای i = 0، یک‌مرتبه اجرا می‌شود، برای i=1، دومرتبه اجرا خواهد شد و برای i=m، m+1 مرتبه اجرا می‌گردد. پس، تعداد تکرار حلقه با شمارنده j به‌صورت زیر حساب می‌شود:

$$1 + \ 2 + \cdots m + (m+1) = \frac{(m+1)(m+2)}{2}$$

چون، حلقه با شمارنده k نیز m مرتبه اجرا می‌شود. پس تعداد تکرار دستور x+=1 برابر است با:

$$\frac{(m+1)(m+2)m}{2}$$

**مثال۱۶. درجه قطعه برنامه زیر چند است؟**

```
for i  in range(n + 1):
  j = i
  while  j != 0:
    j =   j // 2
```

چون قطعه برنامه از دو حلقه متداخل for و while مستقل تشکیل‌شده است، مرتبه اجرائی هر یک از حلقه‌ها را به دست می‌آوریم و در هم ضرب می‌کنیم. یعنی، داریم:

$$\text{مرتبه اجرائی } for = O(n+1) = O(n) + O(1) = O(n) \quad (\text{n + 1بار اجرا می‌شود})$$

$$\text{مرتبه اجرائی } while = O(log_2^n)$$

چون، j در بدترین حالت برابر n است و هر بار j بر ۲ تقسیم می‌گردد. پس مرتبه اجرائی کل برنامه برابر با $O(nlog_2^n)$ می‌باشد.

**مثال۱۷. مرتبه زمانی قطعه برنامه زیر چیست؟**

```
j = n
while  j  >=  1:
    for i  in range(1, n + 1):
        x = x + 1
    j = j // 2
```

مرتبه اجرائی این قطعه برنامه نیز برابر با مرتبه اجرائی while ضرب در مرتبه اجرای for می‌باشد. مرتبه اجرائی while در بدترین حالت برابر با $O(log_2^n)$ است و مرتبه اجرائی for برابر با O(n) می‌باشد. بنابراین، مرتبه اجرائی این قطعه برنامه برابر با $O(nlog_2^n)$ است.

**مثال۱۸. پیچیدگی زمانی دستورات زیر را به دست آورید:**

```
a = n
while  a > 1:
   a = a // 2
   b = n
```

```
while  b > 1:
    b =   b // 3
    x = x + 1
```

پیچیدگی زمانی این دستورات برابر با پیچیدگی زمانی while a > 1 * پیچیدگی زمانی while b > 1 است. چون در داخل حلقه داخلی هر مرتبه b تقسیم بر ۳ می‌شود، پیچیدگی زمانی الگوریتم آن $O(log_3^n)$ می‌باشد و در حلقه while a > 1 چون، هر مرتبه a تقسیم بر ۲ می‌شود، پیچیدگی آن $O(log_2^n)$ خواهد شد. پس، پیچیدگی زمانی کل برابر با $O(log_2^n * log_3^n)$ می‌باشد.

**مثال ۱۹. ثابت کنید عبارت زیر برقرار است:**

$7n^2 - 6n + 2 \in \theta(n^2)$

به ازای $\forall n \geq n_0$ به‌طوری‌که $n_0$، $c_2$ و $\exists c_1$ داریم: $C_2 n^2 < 7n^2 - 6n + 2 < C_1 n^2$

این عبارت را تقسیم بر $n^2$ می‌کنیم تا عبارت زیر به دست آید:

$$C_2 < 7 - \frac{6}{n} + \frac{2}{n^2} < C_1$$

در این رابطه به ازای $n_0 = 3$، $C_1 = 9$ و $C_2 = 6$ رابطه مقابل را داریم: $T(n) \in T(n^2)$

**مثال ۲۰. تعیین کنید آیا عبارت زیر برقرار است یا خیر؟**

$n^5 + 14n^3 \in \theta(n^3)$

با توجه به تعریف θ، $\exists c_1, c_2, n_0 \geq 0, \forall n > n_0 \Rightarrow c_1 n^3 \leq n^5 + 14n^3 \leq c_2 n^3$ برقرار است. چون هیچ‌گاه رابطه $n^5 + 14n^3 \leq c_2 n^3$ برقرار نیست؛ زیرا، اگر طرفین را بر $n^3$ تقسیم کنیم، رابطه زیر به دست می‌آید:

$n^2 + 14 \leq c_2 \Rightarrow n^2 \leq c_2 - 14$

که هیچ‌گاه رابطه $n^2 \leq c_2 - 14$ برقرار نیست. پس، رابطه $T(n) \in \theta(n^3)$ نادرست خواهد بود.

**مثال ۲۱. ثابت کنید عبارت زیر برقرار است:**

$n! + 7n^5 \in O(n^n) \Rightarrow T(n) = n! + 7n^5$

رابطه $\exists C, n_0 \geq 0, \forall n > n_0 \Rightarrow T(n) \leq Cn^n$ برقرار است. پس، داریم:

$T(n) = n! + 7n^5 \leq n^n + 7n^5 \leq n^n + 7n^n = 8n^n$

پس به ازای C = 8 رابطه $T(n) \in O(n^n)$ برقرار خواهد بود.

**مثال ۲۲. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، مجموع ارقام آن را برمی‌گرداند.**

```
def  sumDigit(n):
    if  n == 0:
      return 0
    else:
      return   (n % 10 + sumDigit(n // 10))
```

**مثال ۲۳. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، حاصل‌ضرب ارقام فرد آن را برمی‌گرداند.**

```
def   mulOddDigit(n):
    if  n == 0 :
      return 1
    elif n % 10 % 2 == 1 :
      return   (n % 10 * mulOddDigit(int(n / 10)))
    else:
      return   (mulOddDigit(n // 10))
```

**مثال ۲۴. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، حاصل جمع ارقام بالای ۶ آن را برمی‌گرداند.**

```
def sumDigitG6(n):
    if  n == 0 :
      return 0
    elif n % 10 > 6 :
       return   (n % 10 + sumDigitG6(n // 10))
    else:
       return   (sumDigitG6(int(n / 10)))
```

**مثال ۲۵. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، nامین جمله فیبوناچی را برمی‌گرداند.**

```
def Fibo(n):
    if n == 1 or n == 2 :
      return 1
    else:
      return (Fibo(n -1) + Fibo(n – 2))
```

**مثال ۲۶. تابع بازگشتی که m و n را به‌عنوان پارامتر دریافت کرده، $\binom{n}{m} = C(n,\ m)$ را برمی‌گرداند.**

```
def  C(n, m):
    if (n == m or  m == 0):
      return 1
    else:
      return (C(n -1, m -1) + C(n -1, m))
```

**مثال ۲۷. تابع بازگشتی که a و n را به‌عنوان پارامتر دریافت کرده، حاصل عبارت $\sqrt{a+\sqrt{a+\sqrt{a+\cdots}}}$ را برای n مرتبه برمی‌گرداند.**

```
def  S(a, n):
    if (n == 1):
      return (a**0.5)
    else:
      return((a + S(a, n -1)**0.5)
```

**مثال ۲۸. تابع بازگشتی که a و b را به‌عنوان پارامتر دریافت کرده، a تقسیم بر b را برمی‌گرداند.**

```
def Div   (a, b):
    if (a < b):
      return 0
    else:
      return (1 + Div(a – b, b))
```

**مثال ۲۹. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، حاصل جمع اعداد فرد ۱ تا n را برمی‌گرداند.**

```
def sumOdd (n):
    if (n < 1):
      return 0
    elif (n % 2 == 1):
      return (n + sumOdd(n – 2))
  else:
      return (sumOdd(n - 1))
```

**مثال ۳۰. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، حاصل‌ضرب اعداد زوج ۲ تا n را برمی‌گرداند.**

```
def  MulEven(n):
    if (n < 2):
      return 1
    elif (n % 2 == 0):
      return (n * MulEven(n – 2))
  else:
      return (MulEven(n - 1))
```

**مثال ۳۱. تابع بازگشتی که اعداد صحیح a و b بزرگ‌تر از صفر را دریافت کرده، a ضرب در b را برمی‌گرداند.**

```
def  Mul(a, b):
    if (b == 1):
      return a
    else:
      return (a + Mul(a, b – 1))
```

**مثال ۳۲. تابع بازگشتی که a و b (اعداد صحیح) را به‌عنوان پارامتر دریافت کرده، باقی‌مانده تقسیم صحیح a بر b را برمی‌گرداند.**

```
def  Mod(a, b):
    if (a < b):
      return a
    else:
      return(Mod(a – b, b))
```

**مثال ۳۳. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، مجموع تمام اعداد مضرب ۳ کوچک‌تر یا مساوی n را برمی‌گرداند.**

```
def M3(n):
   if (n < 3):
       return 0
   elif (n % 3 == 0):
       return (n + M3(n – 3))
   else:
       return (M3(n – 1))
```

**مثال ۳۴. تابع بازگشتی که n را به‌عنوان پارامتر دریافت کرده، تعداد صفرهای آن را برمی‌گرداند.**

```
def countZero(n):
   if (n == 0):
       return 0
   elif ( n % 10 == 0):
       return (1 + countZero(n // 10))
   else:
       return (countZero(n // 10))
```

**مثال ۳۵. تعداد تکرار حلقه زیر چند است؟**

```
for i in range (a, b + 1, k):
```

تعداد تکرار حلقه برابر با $\left\lfloor \frac{b-a+1}{k} \right\rfloor$ است.

**مثال ۳۶. مرتبه اجرای دستور x = x+ 1 را مشخص نمایید:**

```
for i in range (2, n, 2):
      x = x + 1
```

حلقه $\left\lfloor \frac{n-2}{2} \right\rfloor$ بار اجرا می‌گردد. پس مرتبه الگوریتم o(n) است.

**مثال ۳۷. مرتبه اجرایی دستور x = x +1 را مشخص کنید:**

```
for i in range (0, n **2, 2):
    x = x + 1
```

حلقه $\frac{n^2}{2}$ مرتبه اجرا می‌شود. بنابراین مرتبه اجرایی الگوریتم $o(n^2)$ است.

**مثال ۳۸. مرتبه اجرایی دستور x = x+ 1 را مشخص نمایید:**

```
for i in range (1, n + 1):
    for j in range (1, m + 1):
        x = x + 1
    for k in range (1, 9 + 1):
        x = x + 1
```

مرتبه اجرای for با شمارنده i برابر n است، ولی مرتبه اجرایی for با شمارنده j برابر m می‌باشد، اما، مرتبه اجرایی for با شمارنده k برابر p است. ازآنجایی‌که حلقه‌ها با شمارنده‌های j و k در داخل حلقه for با شمارنده i قرار دارند و از طرف دیگر، حلقه‌های j و k به‌صورت پشت سر هم هستند (تودرتو نیستند)، پس مرتبه اجرایی آن‌ها با هم جمع شده در n ضرب می‌گردد. بنابراین مرتبه اجرایی کل برابر با $O(n(m + p))$ است.

**پیاده‌سازی ۱-۱. برنامه‌ای که اعمال زیر را انجام می‌دهد:**

- تابع بازگشتی که پارامتر n را دریافت کرده، فاکتوریل n(!n) را برمی‌گرداند (تابع fact).
- تابع بازگشتی که پارامتر n را دریافت کرده، مجموع اعداد فرد ۱ تا n را برمی‌گرداند (تابع SumOdd).
- تابع بازگشتی که پارامتر n را دریافت کرده، مجموع اعداد زوج 1 تا n را برمی‌گرداند (تابع SumEven).
- تابع بازگشتی که پارامتر n را دریافت کرده، مجموع ارقام n را برمی‌گرداند (تابع SumDigits).
- تابع بازگشتی که پارامتر n را دریافت کرده، حاصل‌ضرب ارقام فرد آن را برمی‌گرداند (تابع MulOddDigits).
- تابع بازگشتی که پارامتر n را دریافت کرده، مجموع ارقام بالای ۶ آن را برمی‌گرداند (تابع SumDigitsG6).
- تابع بازگشتی که پارامتر n را دریافت کرده، n امین عدد فیبوناچی را برمی‌گرداند (تابع Fibo).
- تابع بازگشتی که پارامتر n را دریافت کرده، حاصل‌ضرب اعداد زوج ۱ تا n را برمی‌گرداند (تابع MulEven).
- تابع بازگشتی که پارامتر n را دریافت کرده، مجموع اعداد مضرب ۳ از ۱ تا n را برمی‌گرداند(تابع M3).
- تابع بازگشتی پارامتر که n را دریافت کرده، تعداد ارقام صفر آن را شمارش می‌کند (تابع CountZero).
- تابع بازگشتی که a و b را به‌عنوان پارامتر دریافت کرده، $a^b$ را برمی‌گرداند (تابع Pow).
- تابع بازگشتی که پارامترهای a و b را دریافت کرده، حاصل a تقسیم بر b را برمی‌گرداند (تابع Div).
- تابع بازگشتی که پارامترهای a و b را دریافت کرده، حاصل باقی‌مانده تقسیم صحیح a بر b را برمی‌گرداند (تابع Mod).
- تابع بازگشتی که پارامترهای a و b را دریافت کرده، حاصل a ضرب در b را برمی‌گرداند (تابع Mul).
- تابع بازگشتی که a و b را به‌عنوان پارامتر دریافت کرده، حاصل عبارت $\sqrt{a+\sqrt{a+\sqrt{a+\cdots}}}$ (n بار) را برمی‌گرداند (تابع S).

✪ تابع بازگشتی که پارامترهای n و m را دریافت کرده، حاصل $C(^{n}_{m})$ را برمی‌گرداند (تابع C).این توابع بازگشتی توسط تابع main آزمایش شده‌اند.

**مراحل طراحی و اجرا:**

۱. پروژه جدیدی

۲. کرده، دستورات آن را به‌صورت زیر تغییر دهید:

```
def Fact(n):
    if (n == 0):
        return 1
    else:
        return (n * Fact (n - 1))
def sumOdd(n):
    if (n <= 0):
        return 0
    elif (n % 2 == 1):
        return (n + sumOdd (n - 2))
    else:
        return sumOdd (n - 1)
def sumEven(n):
    if (n == 0):
        return 0
    elif (n % 2 == 0):
        return (n + sumEven (n - 2))
    else:
        return sumEven (n - 1)
def sumDigits(n):
    if (n == 0):
        return 0
    else:
        return (n % 10 + sumDigits (int(n / 10)))
def mulOddDigit(n):
    if (n == 0):
        return 1
    elif (n % 10 % 2 == 1):
        return (n % 10 * mulOddDigit (int(n / 10)))
    else:
        return (mulOddDigit (int(n / 10)))
def sumDigitG6(n):
    if (n == 0):
        return 0
    elif (n % 10 > 6):
        return (n % 10 + sumDigitG6 (int(n / 10)))
    else:
        return (sumDigitG6 (int(n / 10)))
def Fibo(n):
    if (n == 1 or n == 2):
        return 1
    else:
        return (Fibo (n - 1) + Fibo (n - 2))
def mulEven(n):
    if (n < 2):
        return 1
    elif (n % 2 == 0):
        return (n * mulEven (n - 2))
    else:
        return (mulEven (n - 1))
def M3(n):
```

```
    if (n < 3):
        return 0
    elif (n % 3 == 0):
        return (n + M3 (n - 3))
    else:
        return (M3 (n - 1))
def countZero(n):
    if (n == 0):
        return 0
    elif (n % 10 == 0):
        return (1 + countZero (int(n / 10)))
    else:
        return (countZero (int(n / 10)))
def Pow(a, b):
    if (b == 0):
        return 1
    else:
        return (a * Pow (a, b - 1))
def Div(a, b):
    if (a < b):
        return 0
    else:
        return (1 + Div (a - b, b))
def Mod(a, b):
    if (a < b):
        return a
    else:
        return (Mod (a - b, b))
def Mul(a, b):
    if (b == 0):
        return 0
    else:
        return (a + Mul (a, b - 1))
def S(a, n):
    if (n == 1):
        return (a** 0.5)
    else:
        return ((a + S (a, n - 1)**0.5))
def C(n, m):
    if (n == m or m == 0):
        return (1)
    else:
        return (C (n - 1, m - 1) + C (n - 1, m))
n = int (input ("Enter n:"))
print ("Fact(", n, ") = ", Fact (n))
print ( "Sum Odd number 1 .. " , n , " = " , sumOdd (n))
print ( "Sum Even number 1 .. " , n , " = " , sumEven (n))
print ( "Multiply even number Even 2 .. " , n , " = " , mulEven (n))
print ( "Sum 3, 6, ..., " , n , " = " , M3 (n))
print ( "Sum digits " , n , " = " , sumDigits (n))
print ( "Count zero digits " , n , " = " , countZero (n))
print ( "Multiply odd digits " , n , " = " , mulOddDigit (n))
print ( "Sum digits > 6 " , n , " = " , sumDigitG6 (n))
print ( "Fibo(" , n , ") = " , Fibo (n))
a = int (input ("Enter a:"))
b = int (input ("Enter b:"))
print(a , " ^ " , b , " = " , Pow (a, b))
print ( a , " * " , b , " = " , Mul (a, b))
print ( a , " / " , b , " = " , Div (a, b))
print ( a , " % " , b , " = " , Mod (a, b))
```

```
print ( "Sqrt(Sqrt(... (Sqrt(" , a , " ))...) = " , S (a, b))
print ( "C(" , a , "," , b , ") = " , C (a, b))
```

۳. پروژه را ذخیره و اجرا کرده تا نمونه خروجی را به‌صورت زیر ببینید:

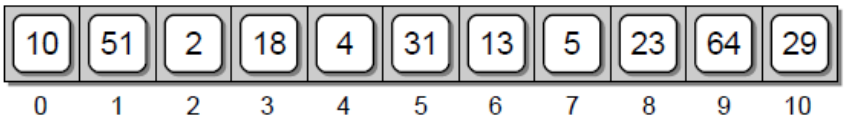
```
Enter n:18
Fact( 18 ) =  6402373705728000
Sum Odd number 1 ..  18  =  81
Sum Even number 1 ..  18  =  90
Multiply even number Even 2 ..  18  =  185794560
Sum 3, 6, ...,  18  =  63
Sum digits  18  =  9
Count zero digits  18  =  0
Multiply odd digits  18  =  1
Sum digits > 6  18  =  8
Fibo( 18 ) =  2584
Enter a:14
Enter b:4
14  ^  4  =  38416
14  *  4  =  56
14  /  4  =  3
14  %  4  =  2
Sqrt(Sqrt(... (Sqrt( 14  ))...) =  18.241672266206855
C( 14 , 4 ) =  1001
```

# فصل ۲ آرایه

آرایه، ساختمان داده‌ای از نوع خطی می‌باشد که شامل مجموعه عناصر هم نوع و هم‌جوار در حافظه بوده و با یک نام ولی اندیس‌های مختلف معرفی می‌شود. هر عنصر آن با **اندیس**[1] و **مقدار**[2] مشخص می‌گردد. از آرایه در مواردی **نظیر لیست خطی یا ترتیبی، ماتریس، چندجمله‌ای، صف و پشته، درخت و گراف، پردازش تکاملی و الگوریتم‌های ژنتیک، شبیه‌سازی، پردازش تصویر و ایجاد فونت** می‌توان استفاده نمود. آرایه‌ها می‌توانند به‌صورت یک‌بعدی، دوبعدی یا چندبعدی استفاده شوند که هرکدام کاربرد خاصی دارند.

## ۱– ۲. آرایه یک‌بعدی( بردار)

از این نوع آرایه برای معرفی لیست‌های خطی، صف، پشته و... استفاده می‌شود که در این فصل و فصل‌های بعدی بررسی می‌شوند. شکل زیر آرایه‌ای یک‌بعدی با ۱۱ عنصر را نمایش می‌دهد:




### ۱–۱– ۲. مفاهیم کلی و پیاده‌سازی آرایه یک‌بعدی

قبل از به کار بردن الگوریتم‌های مربوط به آرایه یک‌بعدی، بهتر است اشاره مختصری به نحوه تعریف و مقداردهی آرایه داشته باشیم.

۱. **تعریف آرایه**، برای تعریف آرایه می‌توانید از کلاس array استفاده کنید. این کلاس در ماژول array قرار دارد. لذا، برای استفاده از این کلاس ابتدا باید ماژول array را با دستور زیر به برنامه اضافه کنید:

```
from array import *
```

اکنون می‌توانید از کلاس آرایه به‌صورت زیر استفاده کنید:

```
array(typecode [, initializer]) -> array
```

این کلاس دو پارامتر را دریافت می‌کند. پارامتر typecode، نوع آرایه را مشخص می‌کند. مقادیری که این پارامتر می‌پذیرد در جدول ۱ – ۲ آمده‌اند و پارامتر [initalizer]، مقادیر اولیه آرایه را تعیین می‌کند.

---

[1]. Index [2] .value

| جدول ۱- ۲ مقادیر پارامتر TypeCode برای تعریف آرایه. | | | | | |
|---|---|---|---|---|---|
| مقدار | نوع C | حداکثر اندازه | مقدار | نوع C | حداکثر اندازه |
| 'b' | عدد صحیح با علامت | ۱ بایت | 'B' | عدد صحیح بدون علامت | ۱ بایت |
| 'u' | کاراکتری یونیکد | ۲ بایت | 'h' | عدد صحیح با علامت | ۲ بایت |
| 'H' | عدد صحیح بدون علامت | ۲ بایت | 'i' | عدد صحیح با علامت | ۲ بایت |
| 'I' | عدد صحیح بدون علامت | ۲ بایت | 'l' | عدد صحیح با علامت | ۴ بایت |
| 'L' | عدد صحیح بدون علامت | ۴ بایت | 'q' | عدد صحیح با علامت | ۸ بایت |
| 'Q' | عدد صحیح بدون علامت | ۸ بایت | 'f' | عدد اعشاری | ۴ بایت |
| 'd' | عدد اعشاری | ۸ بایت | | | |

به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('i', [])
```

دستور اول، ماژول array را به برنامه اضافه می‌کند، دستور دوم، آرایه‌ای به نام a بدون هیچ عضوی با نوع صحیح ۱۶ بیتی تعریف می‌کند.

پس از ایجاد آرایه، اکنون می‌توانید با متدهای آن اعمالی را روی آرایه انجام دهید. برخی از این متدها و خواص کلاس array عبارت‌اند از:

- **متد ()append**، برای اضافه کردن مقداری به انتهای آرایه به کار می‌رود. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('i', [])
>>> a.append(12)
>>> print(a)
```

این دستورات ابتدا، آرایه‌ای به نام a با نوع صحیح ۱۶ بیتی تعریف کرده، ۱۲ را به انتهای آن اضافه می‌کند و دستور آخر مقدار آرایه a (یعنی، array('i', [12]) ) را نمایش می‌دهد.

- **متد ()buffer_info**، تاپلی را برمی‌گرداند که آدرس و طول آرایه می‌باشد.
- **متد ()tobytes**، مقادیر آرایه را به بایت تبدیل می‌کند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('I', [])
>>> for i in range(65, 80):
    a.append(i)
>>> print(a.tobytes())
```

این دستورات، آرایه‌ای به نام a با نوع عددی صحیح بدون علامت تعریف کرده، مقادیر ۶۵ تا ۷۹ به آرایه اضافه می‌کند و در پایان، اعضای آرایه را به بایت تبدیل می‌کند و نمایش می‌دهد(خروجی زیر):

```
b'A\x00\x00\x00B\x00\x00\x00C\x00\x00\x00D\x00\x00\x00E\x00\x00\x00F\
x00\x00\x00G\x00\x00\x00H\x00\x00\x00I\x00\x00\x00J\x00\x00\x00K\x00\x00\
x00L\x00\x00\x00M\x00\x00\x00N\x00\x00\x00O\x00\x00\x00'
```

- **متد ()extend**، یک سری عناصر را به انتهای آرایه اضافه می‌کند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('i', [1, 2, 3])
```

```
>>> a.extend([3, 4])
>>> print(str(a))
```

این دستورات، ابتدا آرایه‌ای با نوع صحیح به نام a با مقادیر اولیه [1, 2, 3] ایجاد کرده، سپس مقادیر [3, 4] را به انتهای آن اضافه کرده و در پایان آرایه a را به‌صورت زیر نمایش می‌دهند:

array('i', [1, 2, 3, 3, 4])

- **متد count()**، تعداد تکرار یک مقدار را در آرایه شمارش می‌کند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('b', [1, 2, 3, 1, 4, 1])
>>> print(a.count(1))
```

این دستورات، ابتدا آرایه‌ای به نام a با نوع بایت با مقادیر اولیه [1, 2, 3, 1, 4, 1] تعریف می‌کنند و در پایان، تعداد عناصر آرایه با مقدار ۱ را شمارش کرده، (یعنی ۳) نمایش می‌دهند.

- **متد index()**، مکان اولین وقوع مقداری را برمی‌گرداند. اگر مقدار در آرایه موجود نباشد، یک پیغام خطا صادر خواهد شد. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('b', [1, 2, 3, 1, 4, 2])
>>> print(a.index(2))
>>> print(a.index(5))
```

این دستورات، ابتدا آرایه‌ای به نام a با نوع بایت تعریف کرده، مقادیر اولیه [1, 2, 3, 1, 4, 2] را به آن تخصیص می‌دهند، سپس مکان اولین وقوع مقدار ۲ (یعنی، ۱) را نمایش می‌دهند و در پایان، چون مقدار ۵ در آرایه وجود ندارد، پیغام خطای زیر را صادر می‌کنند:

```
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    print(a.index(5))
ValueError: array.index(x): x not in list
```

- **متد insert()**، عنصری را در مکان خاصی از آرایه اضافه می‌کند. این متد دو پارامتر را می‌گیرد، پارامتر اول، مکانی است که مقدار پارامتر دوم باید قبل از آن مکان درج شود. اگر مقدار پارامتر اول، بیش‌تر از تعداد عناصر آرایه باشد، مقدار پارامتر دوم را به انتهای آرایه اضافه می‌کند. دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('B', [1, 2, 3, 1, 4, 2])
>>> a.insert(1, 45)
>>> print(a)
>>> a.insert(10, 45)
>>> print(a)
```

این دستورات، ابتدا آرایه a را با نوع بایت و بدون علامت تعریف می‌کنند، سپس مقادیر [1, 2, 3, 1, 4, 2] را به آن تخصیص می‌دهند، در ادامه، مقدار ۴۵ را به قبل از مقدار ۲ (اندیس یک) اضافه کرده، آرایه a (یعنی array('B', [1, 45, 2, 3, 1, 4, 2]) ) را نمایش می‌دهند و در پایان، مقدار ۴۵ را به انتهای آرایه a اضافه نموده و آرایه a (یعنی array('B', [1, 45, 2, 3, 1, 4, 2, 45]) ) را نمایش می‌دهند.

**متد ()remove**، مقداری را از آرایه حذف می‌کند (اولین وقوع مقدار را از آرایه حذف می‌کند). اگر مقدار در آرایه وجود نداشته باشد، یک پیغام خطا صادر خواهد شد. به‌عنوان‌مثال، دستورات زیر را در نظر بگیرید:

```
>>> from array import *
>>> a = array('d', [1, 2, 3, 1])
>>> a.remove(1)
>>> print(a)
>>> a.remove(4)
```

این دستورات، ابتدا آرایه‌ای به نام a با نوع اعشاری با دقت مضاعف تعریف کرده، مقادیر [1, 2, 3, 1] را به اعضای آن تخصیص می‌دهند، سپس اولین مقدار ۱ را حذف کرده، آرایه a (یعنی array('d', [2.0, 3.0, 1.0]) ) را نمایش می‌دهند. دستور آخر، می‌خواهد مقدار ۴ را از آرایه a حذف کند، چون در آرایه وجود ندارد، پیغام خطای زیر را صادر می‌کند:

```
Traceback (most recent call last):
  File "<pyshell#71>", line 1, in <module>
    a.remove(4)
ValueError: array.remove(x): x not in list
```

**متد ()pop**، مقدار عنصری از آرایه را با توجه به اندیس آن برمی‌گرداند و آن عنصر را از آرایه حذف می‌کند. اگر اندیسی که برای برگرداندن عناصر آرایه به کار می‌رود بزرگ‌تر یا مساوی تعداد عناصر آرایه و کوچک‌تر از صفر باشد، یک پیغام خطا صادر می‌شود. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('d', [1, 2, 3, 4])
>>> print(a.pop(2))
>>> print(a.pop(5))
>>> print(a)
```

دستور دوم، آرایه‌ای به نام a با نوع عدد اعشاری با دقت مضاعف تعریف کرده، مقادیر [1.0, 2.0, 3.0, 4.0] را به آن تخصیص می‌دهد. دستور سوم، عنصر سوم (یعنی، عنصر با اندیس ۲ ) آرایه را برمی‌گرداند و از آرایه حذف می‌نماید (مقدار 3.0 را نمایش می‌دهد)، دستور چهارم، می‌خواهد عنصری با اندیس ۵ را برگرداند و از آرایه حذف کند، چون این عنصر در آن وجود ندارد، پیغام خطای زیر را صادر می‌نماید:

```
Traceback (most recent call last):
  File "<pyshell#75>", line 1, in <module>
    print(a.pop(5))
IndexError: pop index out of range
```

در پایان، عناصر آرایه a را نمایش می‌دهد(خروجی زیر):

```
array('d', [1.0, 2.0, 4.0])
```

**متد ()reverse**، عناصر آرایه را معکوس می‌کند. به‌عنوان‌مثال، دستورات زیر را در نظر بگیرید:

```
>>> from array import *
>>> a = array('d', [1, 2, 3, 4])
>>> a.reverse()
>>> print(a)
```

این دستورات، آرایه‌ای به نام a با نوع اعشاری با دقت مضاعف و مقادیر اولیه [1.0, 2.0, 3.0, 4.0] تعریف کرده، آن را معکوس می‌نمایند و نمایش می‌دهند(خروجی زیر):

```
array('d', [4.0, 3.0, 2.0, 1.0])
```

- **خاصیت itemsize**، اندازه هر عنصر آرایه را برمی‌گرداند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('d', [1, 2, 3, 4])
>>> print(a.itemsize)
```

این دستورات، آرایه‌ای به نام a با نوع اعداد اعشاری با دقت مضاعف و مقادیر [1.0, 2.0, 3.0, 4.0] تعریف کرده، اندازه هر عنصر آرایه a ( یعنی، ۸ ) را نمایش می‌دهند.

- **خاصیت typecode**، مقدار typecode آرایه‌ای را برمی‌گرداند. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
>>> from array import *
>>> a = array('I', [1, 2, 3, 4])
>>> print(str(a.typecode))
```

این دستورات، آرایه‌ای به نام a با نوع عددی صحیح ۱۶ بیتی بدون علامت تعریف کرده، مقادیر [1, 2, 3, 4 را به آن تخصیص می‌دهند، در پایان، مقدار typecode آرایه (یعنی، همان I) را نمایش می‌دهند.

## ۴-۱-۲. دسترسی به عناصر آرایه

همان‌طور که بیان گردید، برای دسترسی به عناصر آرایه از اندیس آن به‌صورت زیر استفاده می‌شود:

**[ اندیس] نام آرایه**

اندیس، شماره خانه آرایه را تعیین می‌کند. اندیس آرایه در پایتون از صفر شروع می‌شود. بنابراین، حداکثر مقداری که اندیس می‌تواند بپذیرد، برابر با **[۱- تعداد عناصر آرایه]** است. یعنی، آرایه‌ای با ۵ عنصر، به‌صورت زیر نمایش داده می‌شود:

| a[0] | a[1] | a[2] | a[3] | a[4] |
|---|---|---|---|---|

همان‌طور که در شکل می‌بینید، آخرین خانه آرایه دارای اندیس ۴ است. اکنون دستورات زیر را ببینید:

دستور اول، عنصر سوم آرایه (a[2]) را می‌خواند و دستور دوم، مقدار عنصر سوم آرایه (a[2]) را

```
a[2] = int (input("Enter a number:"))
print (a[2])
```

نمایش می‌دهد.

## ۴-۱-۳. مقداردهی به عناصر آرایه

به دو روش می‌توان به عناصر آرایه مقدار داد. این روش‌ها عبارت‌اند از:

**مقداردهی اولیه به آرایه در هنگام تعریف آن**

اگر بخواهید به آرایه‌ای در هنگام تعریف مقدار اولیه تخصیص دهید، به‌صورت زیر عمل می‌شود:

**مقداردهی به عناصر آرایه با حلقه تکرار و دستور ورودی**

```
>>> a = array('I', [10, 12, 13, 14])
```

مقادیر عناصر آرایه را می‌توان با حلقه‌های تکرار نظیر for، while و دستور ورودی تعیین کرد. در این صورت باید یک حلقه for ایجاد نمود که بدنه آن شامل دستور یا دستوراتی برای تخصیص مقادیر به عناصر آرایه باشد. به‌عنوان‌مثال، دستورات زیر را در نظر بگیرید:

```
a = array ('I', [])
for i  in range (0, 5):
    x = int(input("Enter x:"))
    a = append(x)
```

این دستورات چهار مقدار را خوانده و در خانه‌های تعریف‌شده، دستور اول قرار می‌دهند (در آرایه قرار می‌دهند).

## ۴-۱ - ۴. نمایش عناصر آرایه

برای نمایش عناصر آرایه سه روش وجود دارد که عبارت‌اند از:

**نمایش مقادیر هر عنصر به‌صورت مجزا**

در این روش، با تابع ()print می‌توان مقادیر عناصر آرایه را نمایش داد. به‌عنوان‌مثال، دستورات زیر را ببینید:

```
a = array ('i', [3, 4, 2])
print (a[2], a[1], a[0])
```

دستور اول، آرایه‌ای با سه عنصر تعریف کرده، مقادیر ۳، ۴ و ۲ را به ترتیب به خانه‌های ۰، ۱ و ۲ آن تخصیص می‌دهد و دستور دوم، عناصر آرایه را به‌صورت معکوس (از آخرین به اولین خانه) نمایش می‌دهد.

**نمایش عناصر آرایه با حلقه تکرار**

عناصر آرایه را می‌توان با حلقه‌های تکرار نظیر for، while و تابع ()print نمایش داد. به‌عنوان‌مثال، دستورات زیر را مشاهده کنید:

```
a = array ('i', [10, 5, 7, 8, 0])
 for i  in a:
     print (i, end = '\t')
```

دستور اول، آرایه‌ای با ۵ عنصر به نام a را تعریف کرده، مقادیر ۱۰، ۵، ۷، ۸ و ۰ را به خانه‌های ۰، ۱، ۲، ۳ و ۴ آن نسبت می‌دهد و دستور دوم، با استفاده از حلقه for و تابع ()print مقادیر خانه‌های آرایه را به‌صورت زیر نمایش می‌دهد:

```
10      5       7       8       0
```

**نمایش تمام عناصر آرایه به‌صورت یکجا**

برای نمایش تمام عناصر آرایه به‌صورت یکجا می‌توان نام آرایه را به‌عنوان پارامتر تابع ()print ارسال کرد. به‌عنوان‌مثال، دستورات زیر آرایه‌ی را تعریف کرده و تمام عناصر آن را یکجا نمایش می‌دهند:

```
>>> a = array ('i', [1, 2, 4, 7])
>>> print (a)
```

**ذخیره و بازیابی یک عنصر از آرایه در زمان ثابتی صورت می‌گیرد.**

۲. **چگونگی ذخیره عناصر آرایه در حافظه**، اگر آدرس شروع آرایه x را در حافظه base(x) در نظر بگیریم، آدرس محل ذخیره عنصرiام به‌صورت زیر محاسبه می‌شود: