پایتون

لیست

fruits = [”پرتقال" ,"موز", "سیب"]

🔷**اضافه کردن عنصر**

1️⃣ **append()**

اضافه کردن یک عنصر به انتهای لیست

```
fruits.append("هلو”)
print(fruits)
```

خروجی: ['سیب', 'موز', 'پرتقال', 'هلو']

2️⃣ **insert(index, value)**

اضافه کردن عنصر در مکان مشخص

```
fruits.insert(1,”کیوی”)
print(fruits)
```

خروجی: ['سیب', 'کیوی', 'موز', 'پرتقال', 'هلو']

3️⃣ **extend()**

اضافه کردن **چند عنصر همزمان** (ترکیب دو لیست)

```
fruits.extend(["آلبالو" ,"انار”])
print(fruits)
```

🔷حذف عنصر

1️⃣ `remove(value)`

حذف عنصر بر اساس مقدار

```
fruits.remove("موز")
print(fruits)
```

2️⃣ `pop(index)`

حذف عنصر بر اساس ایندکس و **می‌تواند آن را برگرداند**

```
last = fruits.pop()   # حذف آخرین عنصر
print(last)   # 'آلبالو'
print(fruits)
```

3️⃣ `del`

حذف عنصر یا کل لیست

```
del fruits[1]   # حذف ایندکس 1
print(fruits)

del fruits   # حذف کل لیست
```

4️⃣ `clear()`

خالی کردن کل لیست اما خود لیست باقی می‌ماند

```
fruits.clear()
print(fruits)   # []
```

🔷دسترسی به عناصر

```
fruits = ["سیب", "موز", "پرتقال"]
print(fruits[0])    # 'سیب'
print(fruits[-1])   # 'آخرین عنصر پرتقال'
```

دستور شرطی if

🔷 **تعریف**

`if`برای **گرفتن تصمیم در برنامه** استفاده می‌شود.

اگر یک شرط درست (True)باشد، کد داخل آن اجرا می‌شود.

🔷 **ساختار کلی**

`if`شرط:

دستور

🔷 **کاربرد**

- بررسی سن
- بررسی نمره
- بررسی زوج یا فرد بودن
- بررسی ورود کاربر

```
age = 20

if age >= 18:
    print("شما بزرگسال هستید")
```

دستور if - else

🔷تعریف

اگر شرط درست باشد یک کار انجام بده،
و اگر درست نباشد کار دیگری انجام بده.

```
if :شرط
    دستور اول
else:
    دستور دوم
```

```
number = 7

if number % 2 == 0:
    print(" عدد زوج است")
else:
    print(" عدد فرد است")
```

دستور elif

وقتی چند شرط مختلف داریم استفاده می‌شود.

```
if شرط۱ :
دستور
elif شرط2:
دستور
else:
    دستور نهای
```

```
score = 16
if score >= 18:
    print(”عالی ”)
elif score >= 15:
    print("خیلی خوب”)
elif score >= 10:
    print("قبول”)
else:
    print("مردود”)
```

شرط کوتاه

نسخه‌ی خلاصه‌ی if-else در یک خط است.

وقتی فقط می‌خواهیم بین دو مقدار یکی را انتخاب کنیم استفاده می‌شود.

مقدار_اگر_درست if شرط else مقدار_اگر_غلط

```
score = 8
result = "قبول" if score >= 10 else "مردود"
print(result)
```

روند اجرا:

شرط بررسی می‌شود.

اگر True → مقدار اول انتخاب می‌شود.

اگر False → مقدار بعد از else انتخاب می‌شود.

شرط کوتاه فقط برای **دو حالت** مناسب است.

اگر چند حالت داشته باشیم بهتر است از if-elif-else استفاده کنیم.

عملگر منطقی and

وقتی **همه شرطها باید درست باشند**.

فرمول ذهنی:

- True AND True → True
- در غیر این صورت → False

```
x = 15
if x > 10 and x < 20:
    print("عدد بین 10 و 20 است")
```

هر دو شرط باید درست باشند.

عملگر منطقی or

اگر **حداقل یکی از شرطها درست باشد**، کل شرط درست می‌شود.

True OR False → True

False OR True → True

False OR False → False

```
age = 65

if age < 10 or age > 60:
    print(" تخفیف دارد")
```

اگر یکی از دو شرط برقرار باشد، اجرا می‌شود.

عملگر منطقی not

نتیجه شرط را برعکس می‌کند.
True → False
False → True

```
is_raining = False

if not is_raining:
    print("می‌توانیم بیرون برویم")
```

چون باران نمی‌آید، شرط درست می‌شود.

شرط تو در تو

چرا از شرط تو در تو استفاده می‌کنیم؟
وقتی یک تصمیم وابسته به تصمیم قبلی باشد.
یعنی:
«اگر این درست بود، حالا بیا این یکی رو هم بررسی کن.»

```
username = "admin"
password = "1234"

if username == "admin":
    if password == "1234":
        print("ورود موفق")
    else:
        print("رمز اشتباه است")
else:
    print("نام کاربری اشتباه است")
```

pass

دستور passیعنی «فعلاً هیچ کاری انجام نده».

🔷کاربرد

زمانی که ساختار برنامه را نوشتیم ولی هنوز نمی‌خواهیم کدی داخل آن بنویسیم.
در واقع یعنی:«فعلاً هیچ کاری انجام نده، فقط از این قسمت رد شو.»

```
age = 20

if age > 18:
    pass   # بعداً اینجا کد می‌نویسم
else:
    print("نابالغ")
```

The picture can't be displayed.

الگوسازی ساختاری match

برای بررسی چند حالت مختلف یک متغیر استفاده می‌شود.
شبیه switchدر برخی زبان‌ها.

```
match :متغیر
    case :1مقدار
دستور
    case :2مقدار
دستور
    case _:
        دستور پیش‌فرض
```

```
day = 2

match day:
    case 1:
        print("شنبه ")
    case 2:
        print("یکشنبه")
    case _:
        print("نامعتبر")
```

ترکیب چند مقدار در match

علامت | یعنی «یا»

grade = 18

```
match grade:
    case 20 | 19 | 18:
        print("عالی”)
    case 17 | 16:
        print("خوب”)
    case _:
        print(” نیاز به تلاش بیشتر ”)
```

شرط حفاظتی (Guard)در match

اضافه کردن شرط اضافی داخل caseبا
استفاده از if.

🔷**ساختار**

شرط: ifمتغیر case

```
number = 15

match number:
    case n if n % 2 == 0:
        print("زوج”)
    case n if n % 2 != 0:
        print(”فرد”)
```

حلقه while

تا زمانی که شرط درست باشد، کد تکرار می‌شود.

🔷**ساختار**

```
while شرط:
دستور
```

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

اگر مقدار تغییر نکند، حلقه بی‌نهایت می‌شود.

حلقه for

برای تکرار روی مجموعه‌ای از داده‌ها استفاده می‌شود.

🔷**ساختار**

```
for متغیر in مجموعه:
دستور
```

```
names = ["رضا" ,"سارا", "علی"]
for name in names:
    print(name)
```

تابع range()

برای تولید دنباله‌ای از اعداد استفاده می‌شود (معمولاً در حلقه for.)

🔷**حالت‌ها**

حالت اول:

range(5)

خروجی: 0 تا 4

حالت دوم:

range(2, 6)

خروجی: 2 تا 5

حالت سوم:

range(1, 10, 2)

خروجی: 1, 3, 5, 7, 9

range شروع, پایان, گام چیست؟

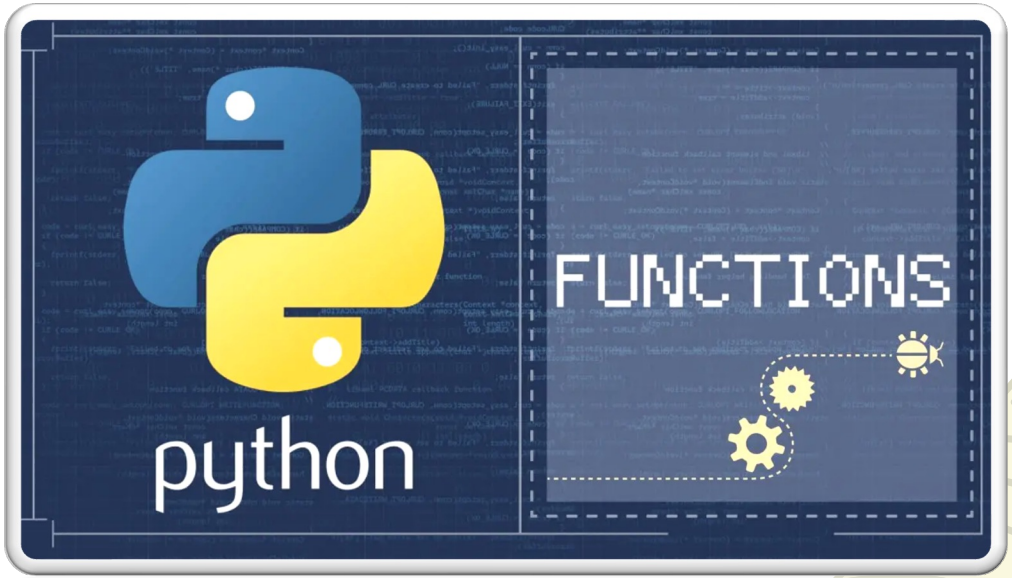
شروع ← عددی که از آن شروع می‌کنیم

پایان ← عددی که می‌خواهیم قبل از آن تمام شود (خودش شامل نمی‌شود)

گام ← فاصله بین اعداد (چقدر زیاد شود در هر مرحله)

```
for i in range(1, 6):
    print("عدد:", i)
```

توابع در پایتون



تابع در پایتون گروهی از عبارت‌های مرتبط است که یک کار مشخص را انجام می‌دهند. توابع کمک می‌کنند تا برنامه به بخش‌های کوچک‌تر و دانه‌بندی شده‌ای شکسته شود.

هرچه برنامه بزرگ و بزرگ‌تر شود، تابع‌ها به سازمان‌یافته‌تر و قابل مدیریت شدن آن کمک می‌کنند. علاوه بر این، توابع مانع از تکرار برنامه‌نویسی برای یک کار واحد می‌شوند و کد را قابل استفاده مجدد می‌کنند.

نحو تابع در پایتون

در ادامه، نحو تابع در پایتون و در واقع، چگونگی نوشتن یک تابع در پایتون آموزش داده شده است.

آنچه در کد زیر نمایش داده شده، تعریف یک تابع است که شامل مولفه‌های زیر می‌شود:

1. کلیدواژه **def** علامت آغاز سرآیند (Header) تابع است.
2. **نام تابع** (function_name) که به صورت یکتا، تابع را مشخص کند. نامگذاری تابع در پایتون از قواعدی مشابه با قواعد نامگذاری شناساگرها (Identifiers) تبعیت می‌کند.
3. **پارامترها (آرگومان‌ها)**که به وسیله آن‌ها، مقادیر به تابع پاس داده می‌شوند. آرگومان‌ها اختیاری هستند.
4. یک **علامت نقل قول** یا **دو نقطه** (:) برای تعیین پایان هدر.
5. داک استرینگ (docstring) اختیاری برای تشریح آنکه تابع چه کاری انجام می‌دهد.
6. یک یا تعداد بیشتری دستور پایتون که بدنه تابع را تشکیل می‌دهند .دستورها باید دارای سطح دندانه‌گذاری یکسانی باشند معمولا ۴ فاصله با کلید tab انجام میشود.
7. یک دستور return اختیاری برای بازگرداندن یک مقدار از تابع.

```
1  def function_name(parameters):
2      """docstring"""
3      statement(s)
4  #مثالی از تابع
5  def greet(name):
6      """This function greets to
7      the person passed in as
8      parameter"""
9      print("Hello, " + name + ". Good morning!")
```

چگونگی فراخوانی یک تابع در پایتون

پس از آنکه یک تابع تعریف شد، می‌توان آن را در تابعی دیگر، برنامه یا حتی «خط فرمان پایتون (Python Prompt)» فراخوانی کرد.
برای فراخوانی یک تابع در پایتون، کافی است که نام آن را با پارامترهای مناسب آن نوشت.

```
>>> greet('Paul’)
Hello, Paul. Good morning!
```

با اجرای کد بالا در شل پایتون، می‌توان خروجی را مشاهده کرد.

داک استرینگ

(Docstring)
اولین رشته پس از عنوان تابع را Docstring می‌گویند. Docstring مخففی برای Documentation String است. از docstring برای ارائه تعریفی کوتاه از عملکرد تابع استفاده می‌شود. اگرچه استفاده از docstring اختیاری است، اما مستندسازی یک کار مهم در برنامه‌نویسی محسوب می‌شود.
در هر شرایطی نیاز به مستندسازی کدها وجود دارد. در مثال بالا، یک docstringبلافاصله بعد از هدر تابع آماده است. معمولا از سه «'» (به صورت ''') برای نوشتن داک‌استرینگ استفاده می‌شود، بنابراین می‌توان آن را تا چند خط ادامه داد. این رشته به عنوان خصیصه __doc__تابع در دسترس خواهد بود. برای مثال، می‌توان کد زیر را در شل پایتون اجرا و خروجی را مشاهده کرد.

```
>>> print(greet.__doc__)
This function greets to
 the person passed into the
 name parameter
```

آرگومان در برنامه نویسی چیست؟

آرگومان یا Argumentمقدار واقعی است که هنگام فراخوانی تابع به آن ارسال می‌شود تا عملیات مشخصی انجام شود.در واقع ارگومان ها بعد از نام تابع و درون پرانتز مشخص میشوند شما میتوانید هر تعداد ارگومان که میخواهید داشته باشید فقط کافی است آن ها را با کاما از هم جدا کنید.

اطلاعات را می‌توان به توابع به عنوان آرگومان‌ها (arguments) داد.آرگومان‌ها بعد از نام تابع و درون پرانتز مشخص می‌شوند. شما می‌توانید هر تعداد آرگومان که بخواهید اضافه کنید، فقط آن‌ها را با کاما از هم جدا کنید. مثال زیر یک تابع با یک آرگومان (fname) نشان می‌دهد. وقتی تابع فراخوانی می‌شود، یک نام اول (first name) به آن داده می‌شود که درون تابع برای چاپ نام کامل استفاده می‌شود:

```
def print_name(fname):
    print("نام شما: " + fname)

print_name("علی")
print_name("سارا")
```

پارامتر در برنامه نویسی چیست و با آرگومان چه تفاوتی دارد؟

پارامتر Parameterبه متغیرهایی گفته می‌شود که در تعریف یک تابع مشخص می‌شوند. این متغیرها مشخص می‌کنند که تابع چه ورودی‌هایی می‌تواند دریافت کند. پارامترها این متغیرها به عنوان ورودی‌های مورد نیاز در تعریف تابع در نظر گرفته می‌شوند و زمانی که تابع فراخوانی می‌شود، آرگومان‌ها جایگزین آن‌ها می‌شوند. به عبارت دیگر، پارامترها مشخص می‌کنند که تابع به چه داده‌هایی نیاز دارد و آرگومان‌ها همان مقادیر واقعی هستند که هنگام اجرای تابع به آن ارسال می‌شوند.

مثلاً در تابع زیر:

```
void PrintMessage(string message) {
    Console.WriteLine(message);
}
```

در این مثال، messageیک پارامتر است. زمانی که این تابع فراخوانی می‌شود، باید مقداری به این پارامتر داده شود که به آن آرگومان می‌گوییم.

تعداد آرگومان‌ها:

به طور پیش‌فرض، یک تابع باید با تعداد صحیحی از آرگومان‌ها فراخوانی شود. این به این معناست که اگر تابع شما ۲ آرگومان انتظار دارد، شما هم باید دقیقاً ۲ آرگومان به آن بدهید نه بیشتر و نه کمتر.

```
def full_name(first, last):
    print("نام کامل:", first, last)

# ست: دو آرگومان داده شده استدر
full_name("رفایی", "علي")
```

تصویر۴۹

آرگومان‌های دلخواه

اگر تعداد آرگومان‌هایی که به تابع شما داده می‌شوند را نمی‌دانید، یک ستاره (*) قبل از نام پارامتر در تعریف تابع اضافه کنید.این روش باعث می‌شود تابع یک تاپل از آرگومان‌ها را دریافت کند و بتواند به عناصر آن به‌درستی دسترسی داشته باشد:

```
def sum_numbers(*args):
    total = sum(args)
    print("مجموع اعداد:", total)

# فراخوانی تابع با تعداد مختلف آرگومان‌ها
sum_numbers(1, 2, 3)          # 6 :خروجی: مجموع اعداد
sum_numbers(10, 20)           # 30 :خروجی: مجموع اعداد
sum_numbers(5, 15, 25, 35) # 80 :خروجی: مجموع اعداد
```

- شما می‌توانید آرگومان‌ها را با استفاده از سینتکس (کلید=مقدار) به تابع بدهید. به این ترتیب، ترتیب آرگومان‌ها دیگر مهم نیست ، چون پایتون می‌داند هر مقدار مربوط به کدام پارامتر است.

```
def my_function(child3, child2, child1):
    print("کوچکترین فرزند:", child3)

# خوانی تابع بافرا keyword arguments
my_function(child1="علي", child2="رضا", child3="ندا")
```

تصویر۵۱

- برای اینکه یک تابع بتواند یک مقدار (یا هر نوع داده‌ای) را به جای خود بازگرداند، از دستور return استفاده می‌کنیم. این مقدار می‌تواند نتیجه یک محاسبه، یک متغیر، یک لیست، یک عدد، یک رشته و غیره باشد.

```
def multiply(x):
    return x * 5

# استفاده از تابع و ذخیره نتیجه در یک متغیر
result = multiply(10)
print("جه ضربنتی:", result)
```

لمبدا در پایتون چیست؟

لمبدا یک روش ساده برای تعریف تابع در پایتون است. این توابع غالباً به نام «عملگرهای لامبدا» یا «تابع‌های لامبدا» نامیده می‌شوند.
اگر قبلاً از پایتون استفاده کرده باشید، احتمالاً توابع خود را با استفاده از کلیدواژه defپایتون تعریف می‌کنید و این روش نیز تاکنون برای شما به خوبی جواب داده است. پس چرا باید از روش دیگری برای تعریف تابع‌ها استفاده کنیم؟

دلیل این مسئله آن است که تابع‌های لامبدا ناشناس هستند. بدین معنی که این‌ها توابعی هستند که لازم نیست نامی برایشان تعیین کنید.
این روش برای تعریف تابع‌های کوچک یک‌بار مصرف در مواردی که تابع اصلی بسیار بزرگ و حجیم است، استفاده می‌شود.

تابع لمبدا به Returnدر پایتون نیاز ندارد. لمبداها می‌توانند هر تعداد آرگومان که لازم باشد داشته باشند؛ اما تنها یک عبارت دارند.
نمی‌توان توابع دیگر را درون یک لامبدا فراخوانی کرد.

رایج‌ترین استفاده از تابع‌های لمبدا در کدهایی است که نیازمند توابع یک‌خطی ساده‌ای هستند و نوشتن یک تابع معمولی کامل، زیاده کاری محسوب می‌شود. این مسئله در ادامه در بخش «نگاشت، فیلتر و کاهش» بیشتر توضیح داده شده است.

چگونه از لمبداها در پایتون استفاده کنیم؟

```
def add_five(number): return number + 5
 print(add_five(number=4))
```

این تابع کاملاً ابتدایی است؛ اما به منظور نمایش کارکرد لامبداها ارائه شده است. تابعی که شما استفاده می‌کنید، ممکن است بسیار پیچیده‌تر از این باشد. این تابع به هر عددی که از طریق پارامتر numberبه آن ارسال می‌شود، 5 واحد اضافه می‌کند.
تابع لامبدای معادل آن چنین است:

```
add_five = lambda number: number + 5
print(add_five(number=4))
```

در این جا به جای استفاده از defاز کلمه lambdaاستفاده شده است. نیازی به کروشه نیست؛ اما کلمات پس از کلیدواژه lambdaبه عنوان پارامتر ایجاد می‌شوند. از علامت دونقطه (:) برای جدا کردن پارامترها و عبارت استفاده می‌شود. در این مورد عبارت به صورت number + 5است.
نیازی به استفاده از کلیدواژه returnنیست؛ چون لامبدا به طور خودکار این کار را برای شما انجام می‌دهد.
در ادامه شیوه ایجاد یک لامبدا با دو آرگومان را می‌بینید:

```
add_numbers_and_five = lambda number1, number2: number1 + number2 + 5
print(add_numbers_and_five(number1=4, number2=3))
```

لامبداهای پایتون به همراه نگاشت، فیلتر و کاهش

کتابخانه اصلی پایتون سه متد به نام‌های نگاشت ،(map) کاهش (reduce) و فیلتر (filter) دارد. این متدها احتمالاً بهترین دلیل استفاده از تابع‌های لامبدا هستند.
تابع نگاشت دو آرگومان می‌گیرد که یک تابع و یک لیست است. این تابع از تابع ورودی استفاده کرده و آن را روی لیست اجرا می‌کند و لیست اصلاح شده را به صورت یک شیء نگاشت (map) باز می‌گرداند. تابع list برای تبدیل مجدد شیء نگاشتِ حاصل به یک لیست، مورد استفاده قرار می‌گیرد.

- استفاده با map() به‌روزرسانی همه عناصر یک لیست

```
numbers = [1, 2, 3]
doubled = list(map(lambda x: x * 2, numbers))

# خروجی: [2, 4, 6]
```

- استفاده با filter() : فیلتر کردن عناصری که شرط مشخصی را دارند

```
numbers = [2, 5, 8, 10]
filtered = list(filter(lambda x: x > 5, numbers))
# خروجی: [8, 10]
```

تصویر۵۵

- مرتب‌سازی با sorted() یا list.sort() : مرتب کردن لیست بر اساس یک قانون خاص

```
words = ["apple", "hi", "banana"]
sorted_words = sorted(words, key=lambda word: len(word))
# خروجی: ['hi', 'apple', 'banana']
```

تصویر۵۶

- استفاده در توابع دیگر به عنوان آرگومان : زمانی که یک تابع دیگر به یک تابع کوچک نیاز دارد.

```
points = [(1, 2), (3, 4), (5, 1)]
closest = max(points, key=lambda point: point[0])
# خروجی: (5, 1)
```