

سؤال تشریحی مصاحبه پایتون با پاسخ

فهرست مطالب

۱	راهنمای استفاده و دامنه پرسش‌ها
۱	بخش ۱: مبانی زبان و انواع داده؛ پرسش‌های ۱ تا ۲۰
۲	بخش ۲: ساختارهای داده و کار با مجموعه‌ها؛ پرسش‌های ۲۱ تا ۴۰
۳	بخش ۳: توابع، ماژول‌ها و مدیریت فایل؛ پرسش‌های ۴۱ تا ۶۰
۴	بخش ۴: شیء‌گرایی و طراحی کلاس؛ پرسش‌های ۶۱ تا ۸۰
۵	بخش ۵: استثناها، iteratorها، generatorها و کیفیت کد؛ پرسش‌های ۸۱ تا ۱۰۰
۶	منابع

۱ راهنمای استفاده و دامنه پرسش‌ها

این راهنما ۱۰۰ پرسش تشریحی مصاحبه پایتون را با پاسخ‌هایی فشرده و کاربردی پوشش می‌دهد؛ از مبانی و ساختارهای داده تا توابع، شیء‌گرایی، استثناها و generatorها. پرسش‌ها برای مرور پیش از مصاحبه طراحی شده‌اند و پاسخ‌ها در کنار تعریف، تفاوت‌ها و نکته‌های مهم هر مفهوم را روشن می‌کنند. برای آمادگی بهتر، مثال‌ها را در مفسر اجرا کنید و تلاش کنید مفاهیم را با بیان خود توضیح دهید.

۲ بخش ۱: مبانی زبان و انواع داده؛ پرسش‌های ۱ تا ۲۰

۱. پایتون چه نوع زبانی است؟ پایتون یک زبان برنامه‌نویسی سطح بالا، تفسیری (interpreted)، پویانوع (dynamically typed) و چندپارادایمی است که از شیء‌گرایی، برنامه‌نویسی تابعی و دستوری پشتیبانی می‌کند. یعنی کد بدون نیاز به کامپایل مستقیم توسط مفسر اجرا می‌شود و نوع متغیرها در زمان اجرا تعیین می‌گردد، نه در زمان نوشتن کد.
۲. تفاوت list و tuple چیست؟ هر دو نوع داده «ترتیبی» (sequence) هستند و از ایندکس‌گذاری پشتیبانی می‌کنند؛ اما لیست تغییرپذیر (mutable) است و می‌توان عناصر آن را افزود، حذف یا جایگزین کرد، در حالی که تاپل تغییرناپذیر (immutable) است و پس از ساخته شدن دیگر قابل ویرایش نیست. به همین دلیل تاپل به عنوان کلید دیکشنری قابل استفاده است ولی لیست نه.
۳. متغیر در پایتون چگونه کار می‌کند؟ متغیر در پایتون یک «نام» (name) است که به یک شیء در حافظه اشاره می‌کند، نه یک خانه حافظه با نوع ثابت. به همین دلیل چند متغیر می‌توانند به یک شیء واحد اشاره کنند (aliasing) و نوع متغیر با تغییر شیء اشاره شده تغییر کند.
۴. تفاوت mutable و immutable با مثال توضیح دهید. انواع تغییرپذیر مانند dict، list و set اجازه تغییر محتوا پس از ایجاد را می‌دهند؛ a.append(3)؛ a = [1,2] باعث تغییر همان شیء می‌شود. انواع تغییرناپذیر مانند int، float، str و tuple اجازه تغییر ندارند؛ هر «تغییر» در آن‌ها در واقع ساختن یک شیء جدید است، مثلاً s = "ab"؛ s += "c" یک رشته جدید می‌سازد.
۵. تفاوت is و == چیست؟ اپراتور == برابری مقداری (value equality) را بررسی می‌کند، یعنی فراخوانی متد __eq__ دو شیء. اما is هویت (identity) را بررسی می‌کند، یعنی اینکه آیا دو نام به یک شیء واحد در حافظه اشاره می‌کنند یا نه. به همین دلیل دو لیست با محتوای یکسان با == برابرند ولی با is برابر نیستند.
۶. انواع عددی در پایتون کدام‌اند؟ سه نوع عددی اصلی وجود دارد: int (عدد صحیح با دقت نامحدود)، float (عدد اعشاری دوباره دقت IEEE 754 و complex (عدد مختلط). همچنین bool زیرنوعی از int است و True معادل ۱ و False معادل ۰ در نظر گرفته می‌شود.
۷. تفاوت نوع رشته (str) در پایتون ۲ و ۳ چیست؟ در پایتون ۳ نوع str بر مبنای یونیکد است و هر کاراکتر می‌تواند از هر زبانی باشد؛ نوع bytes برای داده‌های باینری جدا شده است. در پایتون ۲ نوع پیش فرض رشته ASCII بود و نوع جداگانه unicode برای متن یونیکد وجود داشت. پایتون ۲ دیگر پشتیبانی نمی‌شود.
۸. None چیست و چه کاربردی دارد؟ None یک شیء یکتا از نوع NoneType است که نمایانگر «نبود مقدار» یا مقدار پیش فرض یک متغیر تعریف نشده است. توابعی که مقدار بازگشتی ندارند در واقع None برمی‌گردانند. مقایسه با None باید با is انجام شود، نه ==، چون None یک شیء singleton است.

۹. Slicing یا برش چیست و چگونه کار می‌کند؟ برش روش استخراج زیرمجموعه از انواع ترتیبی است با الگوی `sequence[start:stop:step]`. ایندکس شروع شامل است و ایندکس پایان خیر؛ بنابراین `a[1:4]` عناصر ایندکس ۱ تا ۳ را می‌دهد. گام منفی باعث رفتن از انتها به ابتدا می‌شود، مثل `a[::-1]` برای معکوس کردن.

۱۰. شناور بودن (floating point) چه محدودیتی دارد؟ اعداد اعشاری در پایتون به صورت دوباره دقت IEEE 754 ذخیره می‌شوند و به علت نمایش باینری، برخی اعداد مانند ۱.۰ دقیقاً قابل نمایش نیستند؛ به همین دلیل `0.1 + 0.2 == 0.3` نتیجه False می‌دهد. برای محاسبات مالی یا دقت بالا از ماژول `decimal` استفاده می‌شود.

۱۱. تفاوت `integer division` و `float division` چیست؟ عملگر `/` همیشه نتیجه اعشاری می‌دهد (در پایتون ۳) و عملگر `//` تقسیم صحیح انجام می‌دهد که قسمت صحیح پایین‌تر را می‌گیرد: `7/2` برابر ۵.۳ و `7//2` برابر ۳ است. برای اعداد منفی، `//` به سمت منفی بی‌نهایت گرد می‌کند، مثل `-4 == 7// -2`.

۱۲. اپراتورهای منطقی در پایتون کدام‌اند و چه رفتاری دارند؟ سه اپراتور `and`، `or` و `not` وجود دارند که short-circuit هستند؛ یعنی ارزیابی را به محض تعیین نتیجه متوقف می‌کنند. علاوه بر این، `and` و `or` یکی از عملوندها را برمی‌گردانند، نه لزوماً True یا False؛ مثلاً `or` برابر "hi" است.

۱۳. تابع `range` چه کاری انجام می‌دهد؟ `range(start, stop, step)` یک شیء ترتیبی تولید می‌کند که دنباله‌ای از اعداد صحیح را به صورت تنبل (lazy) تولید می‌کند، بدون اینکه همه آن‌ها را در حافظه نگه دارد. این شیء تکرارپذیر (iterable) است و در حلقه‌ها بسیار کاربرد دارد، مثل `range(10)` برای اعداد ۰ تا ۹.

۱۴. تفاوت `for` و `while` چیست؟ حلقه `for` برای تکرار روی یک شیء تکرارپذیر (iterable) است و تعداد تکرار معمولاً از پیش مشخص است. حلقه `while` تا زمانی که شرط برقرار باشد تکرار می‌شود و برای حالت‌هایی مناسب است که تعداد تکرار وابسته به رویداد یا ورودی کاربر است.

۱۵. دستور `break` و `continue` چه تفاوتی دارند؟ `break` حلقه را به طور کامل قطع می‌کند و اجرا از اولین خط بعد از حلقه ادامه می‌یابد؛ اما `continue` فقط تکرار جاری را رد می‌کند و به تکرار بعدی حلقه می‌رود. هر دو می‌توانند در `for` و `while` استفاده شوند.

۱۶. دستور `pass` چه کاربردی دارد؟ `pass` یک عملیات خنثی (no-op) است که هیچ کاری انجام نمی‌دهد اما از نظر نحوی لازم است تا بلاک خالی خطای `SyntaxError` ندهد. مثلاً وقتی بعداً قرار است کد نوشته شود یا در تعریف اولیه یک کلاس/تابع خالی استفاده می‌شود.

۱۷. نحوه تعریف تابع در پایتون چگونه است؟ با کلیدواژه `def` سپس نام تابع، پارامترها داخل پرانتز، و بلاک بدنه تابع با تورفتگی. مثال:

```
def greet(name):
    """Return a greeting."""
    return f"Hello, {name}!"
```

یک تابع بدون `return` به طور خودکار `None` برمی‌گرداند.

۱۸. تفاوت مقدار بازگشتی `return` و `print` چیست؟ `return` مقداری را از تابع به فراخواننده برمی‌گرداند تا در ادامه محاسبات استفاده شود؛ `print` فقط چیزی را روی خروجی نمایش می‌دهد و مقدار بازگشتی آن `None` است. یک تابع که فقط `print` می‌کند هیچ مقدار قابل استفاده‌ای به فراخواننده نمی‌دهد.

۱۹. Type Hints یا Type Annotations چیست؟ از پایتون ۳.۵ امکان حاشیه‌نویسی نوع (type annotation) اضافه شد که به صورت اختیاری نوع پارامترها و بازگشت تابع را مشخص می‌کند، مثل `def add(a: int, b: int) -> int`. این حاشیه‌نویسی در زمان اجرا اعمال نمی‌شود اما برای ابزارهای بررسی نوع استاتیک مانند `mypy` و مستندسازی کد بسیار مفید است.

۲۰. `comment` و `docstring` چه تفاوتی دارند؟ کامنت با `#` شروع می‌شود و توسط مفسر کاملاً نادیده گرفته می‌شود؛ فقط برای خوانایی کد انسان است. `docstring` یک رشته مستندسازی است که در ابتدای ماژول، کلاس یا تابع قرار می‌گیرد و توسط مفسر نگهداری می‌شود و از طریق ویژگی `__doc__` در دسترس است؛ ابزارهایی مثل `help()` از آن استفاده می‌کنند.

۳ بخش ۲: ساختارهای داده و کار با مجموعه‌ها؛ پرسش‌های ۲۱ تا ۴۰

۲۱. انواع داده ساختاری در پایتون کدام‌اند؟ چهار ساختار اصلی: `list` (ترتیبی، تغییرپذیر)، `tuple` (ترتیبی، تغییرناپذیر)، `set` (نامرتب، بدون تکرار، تغییرپذیر) و `dict` (نگاشت کلید به مقدار، تغییرپذیر). علاوه بر این‌ها، در ماژول `collections` ساختارهای دیگری مانند `OrderedDict` و `Counter` وجود دارد.

۲۲. چرا `list` به عنوان کلید دیکشنری قابل استفاده نیست؟ کلید دیکشنری باید hashable باشد؛ یعنی مقدار `hash` آن در طول عمر شیء ثابت بماند. لیست تغییرپذیر است و اگر محتوایش تغییر کند، `hash` آن هم تغییر می‌کند و جای عنصر در دیکشنری گم می‌شود. در مقابل `tuple` تغییرناپذیر است و (تا زمانی که عناصر داخلی آن hashable باشند) می‌تواند کلید باشد.

۲۳. تفاوت `append` و `extend` چیست؟ `append(x)` شیء `x` را به عنوان یک عنصر به انتهای لیست اضافه می‌کند؛ در حالی که `extend(iterable)` همه عناصر آن شیء تکرارپذیر را یکی‌یکی به انتها اضافه می‌کند. مثلاً `a=[1,2]; a.append([3,4])` نتیجه `[1,2,[3,4]]` می‌دهد اما `a.extend([3,4])` نتیجه `[1,2,3,4]` است.

۲۴. تفاوت `append` و `insert` چیست؟ `append` همیشه عنصر را به انتهای لیست اضافه می‌کند، اما `insert(index, value)` عنصر را در ایندکس مشخصی درج می‌کند و عناصر بعدی را به سمت راست جابه‌جا می‌کند. مثلاً `a.insert(0, x)` عنصر را در ابتدای لیست قرار می‌دهد؛ این عملیات در لیست پرهزینه است.

۲۵. تفاوت `remove` و `pop` و `del` چیست؟ `remove(value)` اولین عنصری که مقدار آن برابر `value` است را حذف می‌کند و اگر نباشد `ValueError` می‌دهد. `pop(index)` عنصر در ایندکس مشخص را حذف و همان را برمی‌گرداند (پیش‌فرض آخرین عنصر).

دستور del بر اساس ایندکس یا برش حذف می‌کند و هیچ مقداری برنمی‌گرداند؛ حتی می‌تواند کل متغیر را حذف کند.

۲۶. sorted() و list.sort() چه تفاوتی دارند؟ sorted(iterable) یک لیست جدید مرتب‌شده از عناصر ورودی می‌سازد و شیء اصلی را دست نمی‌زند؛ اما list.sort() لیست را درجا (in-place) مرتب می‌کند و مقدار None برمی‌گرداند. هر دو پارامترهای key و reverse را می‌پذیرند.

۲۷. reversed() و list.reverse() چه تفاوتی دارند؟ reversed(seq) یک iterator معکوس برمی‌گرداند که عنصرها را به صورت تنبیل در اختیار می‌گذارد و شیء اصلی را تغییر نمی‌دهد؛ list.reverse() لیست را درجا معکوس می‌کند و None برمی‌گرداند.

۲۸. shallow copy و deep copy چه تفاوتی دارند؟ کپی سطحی (shallow copy) فقط یک لیست جدید می‌سازد که به همان عناصر اصلی اشاره می‌کند؛ یعنی تغییر عناصر داخلی (اگر تغییرپذیر باشند) در هر دو لیست دیده می‌شود. اما deep copy (از طریق copy.deepcopy) به صورت بازگشتی همه عناصر را هم کپی می‌کند و دو ساختار کاملاً مستقل ایجاد می‌کند.

۲۹. List Comprehension چیست و چه مزیتی دارد؟ روشی مختصر برای ساختن لیست‌ها است که به صورت [expression for item in iterable if condition] نوشته می‌شود و جایگزین خواناتر و معمولاً سریع‌تر حلقه for همراه با append است. مثال:

```
squares = [x**2 for x in range(10) if x % 2 == 0]
```

نتیجه لیست مربع اعداد زوج از ۰ تا ۸ است.

۳۰. تفاوت List Comprehension و Generator Expression چیست؟ هر دو syntax مشابه دارند اما لیست کامپرنشن کل لیست را در حافظه می‌سازد با براکت [...]; در حالی که generator expression با پرانتز (...) یک شیء generator می‌سازد که مقادیر را یکی یکی و تنبیل تولید می‌کند و حافظه کمتری مصرف می‌کند، مناسب دنباله‌های بزرگ.

۳۱. tuple packing و unpacking چیست؟ بسته‌بندی (packing) یعنی جمع کردن چند مقدار در یک tuple: t = 1, 2, 3. بازکردن (unpacking) یعنی اختصاص عناصر tuple به چند متغیر: a, b, c = t. از unpacking برای مبادله دو متغیر بدون متغیر موقت هم استفاده می‌شود: a, b = b, a.

۳۲. set چیست و چه تفاوتی با لیست دارد؟ مجموعه (set) یک ساختار نامرتب از عناصر یکتا و hashable است. برخلاف لیست، ایندکس‌گذاری ندارد و عناصر تکراری را خودکار حذف می‌کند. مجموعه از عملیات ریاضی مانند اجتماع، اشتراک، تفاضل - و تقاطع متقارن پشتیبانی می‌کند.

۳۳. frozenset چیست و چه زمانی استفاده می‌شود؟ frozenset نسخه تغییرناپذیر مجموعه است؛ پس از ساخته شدن نمی‌توان به آن عنصر افزود یا از آن حذف کرد. به همین دلیل hashable است و می‌تواند به عنوان کلید دیکشنری یا عنصر یک set دیگر استفاده شود؛ در حالی که set معمولی چنین اجازه‌ای ندارد.

۳۴. dict چیست و چگونه کار می‌کند؟ دیکشنری یک ساختار نگاشت (mapping) از کلید به مقدار است که از جدول hash برای دسترسی سریع O(1) میانگین به هر عنصر استفاده می‌کند. کلیدها باید hashable باشند و از پایتون ۷.۳ ترتیب درج کلیدها حفظ می‌شود.

۳۵. تفاوت dict[key] و dict.get(key) چیست؟ اگر d[key] وجود نداشته باشد KeyError می‌دهد؛ اما d.get(key) در این حالت None (یا مقدار پیش‌فرض دومین آرگومان) برمی‌گرداند و خطا نمی‌دهد. وقتی می‌خواهید خطا زودتر شناسایی شود از [] و وقتی مقدار غیبت قابل قبول است از get استفاده می‌شود.

۳۶. متد items(), keys() و values() در دیکشنری چه کاری می‌کنند؟ این سه متد نمای‌های (views) پویا از کلیدها، مقادیر و جفت‌های (کلید، مقدار) دیکشنری برمی‌گردانند. این نماها تغییرات دیکشنری را منعکس می‌کنند و برای حلقه‌زنی استفاده می‌شوند؛ مثلاً: for k, v in d.items(): روی همه جفت‌ها تکرار می‌کند.

۳۷. defaultdict و setdefault چه کاربردی دارند؟ هر دو برای مقداردهی پیش‌فرض کلیدهای غایب استفاده می‌شوند. d.setdefault(k, default) اگر کلید وجود نداشت آن را با مقدار default می‌سازد؛ collections.defaultdict(list) به جای آن، هنگام دسترسی به کلید غایب تابع سازنده پیش‌فرض را فراخوانی می‌کند و کد را تمیزتر می‌کند.

۳۸. چگونه می‌توان یک دیکشنری را مرتب‌سازی کرد؟ با استفاده از sorted() روی آیتم‌ها و سپس ساخت دیکشنری جدید:

```
sorted_dict = dict(sorted(d.items(), key=lambda item: item[1]))
```

این کار دیکشنری جدیدی می‌سازد که کلیدها یا مقادیر آن بر اساس معیار انتخابی مرتب شده‌اند.

۳۹. رشته (str) چه متدهای پرکاربردی دارد؟ برخی مهم‌ترین‌ها: strip() برای حذف فاصله‌های ابتدا و انتها، split() برای شکستن رشته به لیست، join() برای بهم‌پیوستن عناصر یک iterable، replace() برای جایگزینی، upper() و lower() برای تغییر حروف، و startswith() و endswith() برای بررسی پیشوند/پسوند.

۴۰. f-string چیست و چه تفاوتی با format دارد؟ f-string (از پایتون ۳.۶) راهی برای درج مستقیم متغیرها و عبارات داخل رشته است: f"{name} is {age} years old". با متد format() هم می‌توان همین کار را کرد اما f-string خواناتر، کوتاه‌تر و معمولاً سریع‌تر است و اجازه درج عبارات محاسباتی را هم می‌دهد: f"{2*3}" نتیجه 6 می‌دهد.

۴ بخش ۳: توابع، ماژول‌ها و مدیریت فایل؛ پرسش‌های ۴۱ تا ۶۰

۴۱. *args و kwargs چه هستند؟ *args آرگومان‌های مکانی نامحدود را به صورت tuple و **kwargs آرگومان‌های کلیدواژه‌ای نامحدود را به صورت dict دریافت می‌کند. این الگو برای تابع‌های عمومی یا wrapper مفید است.

۴۲. مقدار پیش‌فرض پارامتر چه نکته‌ای دارد؟ مقدار پیش‌فرض هنگام تعریف تابع، یک بار محاسبه می‌شود. بنابراین استفاده از لیست یا دیکشنری پیش‌فرض باعث اشتراک همان شیء میان فراخوانی‌ها می‌شود. الگوی امن، None به عنوان مقدار پیش‌فرض و ساخت شیء تازه در بدنه تابع است.

- ۴۳. lambda چیست؟** تابع ناشناس تک‌عبارتی است، مانند `lambda x: x * 2`. برای عملیات کوتاه، مثلاً تعیین معیار مرتب‌سازی با `sorted(items, key=lambda x: x[1])` برای منطق پیچیده بهتر است `def` بنویسیم.
- ۴۴. map و filter چه می‌کنند؟** `map(f, iterable)` تابع `f` را روی عناصر اعمال می‌کند و `iterable` نتایج را می‌دهد؛ `filter(f, iterable)` عناصری را نگه می‌دارد که `f` برایشان درست است. هر دو تبدیل‌اند و در صورت نیاز می‌توان خروجی را با `list()` `materialize` کرد.
- ۴۵. decorator چیست؟** تابعی است که تابع دیگری را می‌گیرد و نسخه‌ای با رفتار افزوده برمی‌گرداند، بی‌آنکه بدنه تابع اصلی تغییر کند. علامت `@decorator` نحو کوتاهی برای اعمال آن است. کاربردهای رایج شامل ثبت رویداد، بررسی دسترسی و اندازه‌گیری زمان اجراست.
- ۴۶.**** چرا `decorator` از `*args` و `kwargs` استفاده می‌کند؟ چون `wrapper` باید بتواند ورودی تابع‌های متفاوت را دریافت و بدون دانستن تعداد و نام آرگومان‌ها به تابع اصلی منتقل کند. در بدنه `wrapper` معمولاً `return func(*args, **kwargs)` نوشته می‌شود.
- ۴۷. functools.wraps چه کاربردی دارد؟** این `decorator` فراداده‌هایی مانند نام، `docstring` و ماژول تابع اصلی را به `wrapper` منتقل می‌کند. بدون آن، ابزارهای دیباگ و مستندسازی معمولاً فقط نام `wrapper` را می‌بینند.
- ۴۸. تابع بازگشتی چیست و چه شرطی دارد؟** تابعی است که خودش را فراخوانی می‌کند. برای پایان یافتن باید شرط پایه (base case) داشته باشد و هر بار به آن نزدیک‌تر شود؛ وگرنه `recursion` بی‌پایان یا عبور از محدودیت عمق رخ می‌دهد. برای مسئله‌های بزرگ، حلقه یا الگوریتم تکراری گاهی مناسب‌تر است.
- ۴۹. تفاوت global، local و nonlocal چیست؟** نام `local` در تابع جاری تعریف می‌شود. `global` در تابع به نام سطح ماژول اشاره می‌کند؛ `nonlocal` در تابع تودرتو به نام تابع بیرونی نزدیک اشاره می‌کند. هر دو اعلان دوم برای انتساب به نام بیرونی لازم‌اند.
- ۵۰. قانون LEGB چیست؟** ترتیب جست‌وجوی نام‌هاست: `Local` (محلی)، `Enclosing` (تابع محصور)، `Global` (سطح ماژول)، `Built-in` (نام‌های داخلی). فهم این ترتیب در تحلیل متغیرهای هم‌نام و خطاهای `scope` کمک می‌کند.
- ۵۱. closure چیست؟** وقتی تابع داخلی به نام‌های تابع بیرونی ارجاع دهد، آن نام‌ها می‌توانند پس از پایان اجرای تابع بیرونی نیز در دسترس تابع داخلی بمانند؛ این سازوکار `closure` است. از آن در `decorator` و ساخت تابع‌های دارای وضعیت استفاده می‌شود.
- ۵۲. first-class function یعنی چه؟** تابع مانند دیگر اشیای پایتون می‌تواند به متغیری نسبت داده شود، آرگومان تابع دیگر باشد، از تابعی بازگردد یا در یک ساختار داده ذخیره شود. این ویژگی پایه‌ای برای `callback`ها و برنامه‌نویسی تابعی است.
- ۵۳. iterable و iterator چه تفاوتی دارند؟** `Iterable` شیئی است که می‌توان از آن `iterator` گرفت، معمولاً با `iter(obj)`. `Iterator` وضعیت پیمایش دارد و با `next()` عنصر بعدی را می‌دهد؛ پس از پایان، `StopIteration` رخ می‌دهد. لیست `iterable` است، درحالی‌که `iterator` حاصل از `iter(list)` پیمایش را پیش می‌برد.
- ۵۴. return و yield چه تفاوتی دارند؟** `return` تابع را خاتمه می‌دهد و مقدار بازمی‌گرداند. `yield` مقدار را تحویل می‌دهد اما وضعیت تابع را حفظ می‌کند تا دفعه بعد از همان نقطه ادامه یابد؛ تابع دارای `yield` یک `generator` می‌سازد.
- ۵۵. ماژول چیست و چگونه وارد می‌شود؟** ماژول فایل پایتونی شامل تعریف‌ها و دستورهایست. با `import math` ماژول وارد و اعضای آن با `math.sqrt()` استفاده می‌شود؛ یا می‌توان با `from math import sqrt` نامی مشخص را وارد کرد.
- ۵۶. چرا معمولاً از from module import * پرهیز می‌شود؟** این روش نام‌های زیادی را به فضای نام جاری وارد می‌کند و ممکن است نام‌ها با هم تداخل کنند یا منشأ آن‌ها نامشخص شود. واردکردن ماژول یا نام‌های مشخص، خوانایی و قابلیت نگهداری را بهتر می‌کند.
- ۵۷. "main" == name چه کاربردی دارد؟** هنگام اجرای مستقیم فایل، مقدار `__name__` برابر `"__main__"` است؛ هنگام `import` شدن، نام ماژول خواهد بود. شرط `if __name__ == "__main__":` اجرای بخش آزمایشی یا نقطه شروع برنامه را به اجرای مستقیم محدود می‌کند.
- ۵۸. تفاوت ماژول و package چیست؟** `package` معمولاً یک فایل پایتون است؛ `package` راهی برای سازمان‌دهی چند ماژول در ساختاری سلسله‌مراتبی است. پکیج‌ها واردسازی‌هایی مانند `from package import module` را ممکن می‌کنند؛ در ساختارهای مدرن، `package`های `namespace` نیز می‌توانند بدون `__init__.py` باشند.
- ۵۹. virtual environment چیست؟** محیط مجازی پوشه‌ای جداگانه برای نصب مفسر و وابستگی‌های یک پروژه است. با `python -m venv .venv` ساخته می‌شود و کمک می‌کند نسخه کتابخانه‌های پروژه‌ها با هم تداخل نکنند. وابستگی‌ها را می‌توان در فایل `requirements.txt` یا `Pipfile` ثبت کرد.
- ۶۰. pip و PyPI چه هستند؟** `pip` ابزار نصب و مدیریت بسته‌های پایتون است؛ معمولاً بسته‌ها را از مخزن عمومی `PyPI` دریافت می‌کند. نمونه دستورها: `python -m pip install requests` و `python -m pip freeze`. برای پروژه‌ها، نسخه وابستگی‌ها باید به شکل قابل بازتولید ثبت و مدیریت شود.

۵ بخش ۴: شیء‌گرایی و طراحی کلاس؛ پرسش‌های ۶۱ تا ۸۰

- ۶۱. کلاس و شیء چه تفاوتی دارند؟** کلاس الگو یا تعریف نوع است و ویژگی‌ها و رفتارهای مشترک را مشخص می‌کند؛ شیء (instance) نمونه‌ای ساخته‌شده از آن کلاس است که وضعیت خودش را دارد. کلاس `Car` می‌تواند ویژگی‌هایی مانند رنگ داشته باشد و هر خودرو شیء جداگانه‌ای از آن باشد.
- ۶۲. چگونه در پایتون کلاس تعریف می‌کنیم؟** با کلیدواژه `class` و بدنه‌ای با تورفتگی. متد ویژه `__init__` هنگام ساخت نمونه برای مقداردهی اولیه فراخوانی می‌شود:

```
class User:
```

```
def __init__(self, name):
    self.name = name
```

۶۳. self چیست؟ self قراردادی است برای ارجاع به نمونه جاری داخل متد نمونه. پایتون نمونه را هنگام فراخوانی متد به طور ضمنی به عنوان آرگومان اول می فرستد؛ نام self اجباری نحوی نیست اما باید از قرارداد رایج پیروی کرد.

۶۴. init سازنده است؟ در گفتار روزمره به آن سازنده می گویند، اما دقیق تر، __init__ نمونه ای را که پیش تر ساخته شده مقداردهی می کند. متد __new__ مسئول ساخت نمونه است و در استفاده های معمولی به ندرت بازنویسی می شود.

۶۵. تفاوت متغیر کلاس و متغیر نمونه چیست؟ متغیر کلاس روی خود کلاس تعریف شده و میان نمونه ها مشترک است؛ متغیر نمونه معمولاً با self.name در __init__ ساخته می شود و مختص همان نمونه است. اگر مقدار مشترکی مانند لیست متغیر کلاس باشد، تغییر آن از یک نمونه در همه نمونه ها دیده می شود.

۶۶. متد نمونه، classmethod و staticmethod چه تفاوتی دارند؟ متد نمونه آرگومان ضمنی self می گیرد و با وضعیت نمونه کار می کند. متد کلاس با classmethod @ آرگومان ضمنی cls می گیرد و برای کار با کلاس یا ساخت نمونه های جایگزین کاربرد دارد. متد ایستا با staticmethod @ آرگومان ضمنی ندارد و تابعی مرتبط با کلاس را در فضای نام آن قرار می دهد.

۶۷. وراثت چیست؟ وراثت به کلاس فرزند اجازه می دهد ویژگی ها و متدهای کلاس پایه را به ارث برد یا بازنویسی کند. این کار استفاده مجدد از کد و بیان رابطه «نوعی از» را ممکن می کند؛ اما وراثت را باید زمانی به کار برد که چنین رابطه ای از نظر دامنه مسئله معنادار باشد.

۶۸. چگونه متد کلاس والد را فراخوانی می کنیم؟ با super() می توان به پیاده سازی کلاس بعدی در ترتیب حل متد (MRO) دسترسی پیدا کرد. روش معمول در سازنده فرزند super().__init__(...) است. این الگو در وراثت چندگانه نیز از زنجیره همکاری کلاس ها پشتیبانی می کند.

۶۹. overriding چیست؟ بازنویسی (overriding) یعنی کلاس فرزند متدی با همان نام متد والد تعریف کند تا رفتار ویژه خود را ارائه دهد. اگر لازم باشد رفتار والد نیز اجرا شود، متد فرزند می تواند از super() استفاده کند.

۷۰. چندوراثتی در پایتون چگونه حل می شود؟ کلاس می تواند بیش از یک کلاس پایه داشته باشد. پایتون برای جست و جوی متدها از ترتیب حل متد (MRO) استفاده می کند که با الگوریتم C3 خطی سازی می شود؛ آن را می توان با ClassName.mro() بررسی کرد. برای جلوگیری از ابهام، ساختار چندوراثتی را ساده نگه دارید.

۷۱. آیا پایتون از encapsulation پشتیبانی می کند؟ پایتون بر قراردادهای تکیه دارد، نه کنترل دسترسی سخت. نامی که با یک زیرخط آغاز می شود، مانند _value، نشانه آن است که عضو داخلی محسوب می شود. نام دو زیرخط آغازنشده در کلاس، مانند __value، به طور محدود تغییرنام (name mangling) می شود تا برخورد ناخواسته با نام کلاس فرزند کاهش یابد؛ این سازوکار امنیتی نیست.

۷۲. property چیست؟ @property اجازه می دهد یک متد مانند ویژگی خوانده شود و دسترسی کنترل شده به مقدار فراهم کند. با setter می توان اعتبارسنجی را هنگام انتساب انجام داد. این روش در صورت نیاز به منطق در آینده، رابط کاربری کلاس را پایدار نگه می دارد.

۷۳. متدهای ویژه یا dunder methods چه هستند؟ متدهایی با دو زیرخط در ابتدا و انتها، مانند __len__ و __str__، هستند که رفتار شیء را با قابلیت های زبان هماهنگ می کنند. مثلاً len(obj) متد __len__() را به طور مفهومی به کار می گیرد. معمولاً آن ها را مستقیم فراخوانی نمی کنیم، بلکه از نحو و توابع داخلی استفاده می کنیم.

۷۴. تفاوت str و repr چیست؟ __str__ نمایش خوانا برای کاربر را می سازد؛ __repr__ نمایش فنی تر و مفید برای دیباگ و توسعه دهنده را فراهم می کند. اگر __str__ موجود نباشد، پایتون معمولاً از __repr__ استفاده می کند.

۷۵. operator overloading چیست؟ با پیاده سازی متدهای ویژه می توان معنای عملگرها را برای کلاس تعریف کرد؛ مثلاً __add__ برای + یا __eq__ برای ==. بازتعریف باید با معنای متعارف عملگر سازگار باشد تا استفاده از کلاس برای دیگران قابل پیش بینی بماند.

۷۶. تفاوت == و is هنگام مقایسه اشیا چیست؟ == برابری مقدار را می سنجد و ممکن است با بازنویسی __eq__ تعریف شود؛ is بررسی می کند که دو نام دقیقاً به یک نمونه اشاره کنند. برای مقایسه با None از is None استفاده کنید.

۷۷. dataclass چیست؟ dataclasses در کتابخانه استاندارد با @dataclass متدهای متداولی مانند __init__ و __repr__ را بر اساس فیلدهای کلاس تولید می کند. برای کلاس هایی که عمدتاً داده نگه می دارند مفید است و همچنان امکان افزودن متدهای سفارشی وجود دارد.

۷۸. slots چیست؟ __slots__ مشخص می کند نمونه های کلاس چه ویژگی هایی می توانند داشته باشند و در بعضی کاربردها فضای نمونه را کاهش می دهد. ممکن است انعطاف افزودن ویژگی های دلخواه را محدود کند و با چندوراثتی یا ارث بری نیازمند توجه باشد؛ بنابراین نباید بدون سنجش نیاز استفاده شود.

۷۹. composition چه تفاوتی با inheritance دارد؟ در وراثت، کلاس فرزند نوعی از کلاس پایه است؛ در ترکیب (composition)، یک شیء از اشیای دیگر برای انجام کار استفاده می کند («دارای»). ترکیب اغلب اتصال کمتری میان اجزا ایجاد می کند و امکان جایگزینی وابستگی ها را آسان تر می سازد.

۸۰. شیء hashable چیست و چرا اهمیت دارد؟ شیء hashable دارای hash پایدار در طول عمرش است و با برابری سازگار عمل می کند؛ از این رو می تواند کلید دیکشنری یا عضو مجموعه باشد. اشیای تغییرپذیری که محتوایشان تغییر می کند معمولاً hashable نیستند. کلاس سفارشی برای hash شدن باید برابری و hash سازگار داشته باشد.

۶ بخش ۵: استثناها، iterator، generator و کیفیت کد؛ پرسش های ۸۱ تا ۱۰۰

۸۱. تفاوت خطای نحوی و استثنا چیست؟ خطای نحوی (SyntaxError) زمانی رخ می دهد که کد با دستور زبان پایتون سازگار نباشد و معمولاً پیش از اجرای آن بخش گزارش می شود. استثنا (exception) هنگام اجرای برنامه رخ می دهد؛ مانند تقسیم بر صفر یا

دسترسی به کلید ناموجود.

۸۲. **try و except چگونه کار می‌کنند؟** کد مستعد خطا در بلوک try قرار می‌گیرد و خطاهای مورد انتظار در except مدیریت می‌شوند. بهتر است نوع استثنا را مشخص کنیم، مثلاً: `except ValueError`؛ گرفتن همه خطاها با `except` ممکن است خطاهای برنامه‌نویسی را پنهان کند.

۸۳. **finally چه کاربردی دارد؟** بلوک finally پس از اجرای try و بخش‌های مرتبط، چه استثنا رخ دهد چه نه، برای پاک‌سازی اجرا می‌شود. برای مدیریت فایل معمولاً with مناسب‌تر است، چون آزادسازی منبع را ایمن‌تر و خواناتر انجام می‌دهد.

۸۴. **else در try-except چه می‌کند؟** بخش else تنها زمانی اجرا می‌شود که بلوک try بدون استثنا تمام شده باشد. گذاشتن کد موفقیت در else مرز کد پرخطر را روشن می‌کند و مانع از آن می‌شود که خطای همان کد به اشتباه به عنوان خطای عملیات try گرفته شود.

۸۵. **چگونه استثنای سفارشی تعریف می‌کنیم؟** معمولاً با ساخت کلاسی که از Exception ارث‌بری کند:

```
class InvalidRecordError(Exception):
    pass
```

سپس در موقعیت مناسب با `raise InvalidRecordError(" ")` آن را بالا می‌اندازیم.

۸۶. **raise چه کاری انجام می‌دهد؟** raise یک استثنا را ایجاد و منتشر می‌کند تا جریان عادی برنامه متوقف شود و مدیریت خطا در سطح بالاتر انجام گیرد. درون یک except، دستور بدون آرگومان raise همان استثنای جاری را با ردپای پشته حفظ‌شده دوباره منتشر می‌کند.

۸۷. **چرا استفاده از with برای فایل توصیه می‌شود؟** مدیر زمینه (context manager) در `with open(...) as f:` فایل را پس از پایان بلوک، حتی هنگام بروز استثنا، می‌بندد. این کار احتمال نشت منبع را کم می‌کند و از الگوی try/finally دستی ساده‌تر است.

۸۸. **generator چیست؟** تابعی که با yield مقدار تولید کند، generator function است؛ فراخوانی آن به جای اجرای کامل، gen-erator object می‌دهد. مقدارها هنگام پیمایش و به ترتیب ساخته می‌شوند، بنابراین برای داده‌های حجیم یا جریان‌های نامتناهی مفید است.

۸۹. **مزیت generator نسبت به لیست چیست؟** generator معمولاً همه نتایج را از پیش در حافظه نگه نمی‌دارد، بلکه هر مقدار را هنگام نیاز تولید می‌کند. این ویژگی مصرف حافظه را کاهش می‌دهد، اما generator یک‌بار مصرف است و برای شروع دوباره باید نمونه تازه‌ای ساخت.

۹۰. **generator expression چیست؟** عبارتی فشرده برای ساخت generator است و از پرانتز استفاده می‌کند: `(x*x for x in range(100))`. برخلاف list comprehension با براکت، مقادیر را تدریجی تولید می‌کند و می‌توان آن را مستقیماً به تابعی مانند `sum()` داد.

۹۱. **StopIteration چیست؟** استثنایی است که iterator با رسیدن به پایان پیمایش بالا می‌اندازد. حلقه for آن را به طور خودکار مدیریت می‌کند و پایان تکرار را تشخیص می‌دهد؛ در کاربرد معمول نیاز نیست این استثنا را دستی مدیریت کنید.

۹۲. **چگونه iterator سفارشی می‌سازیم؟** کلاسی که `__iter__()` را پیاده‌سازی کند و iterator برگرداند و `__next__()` را برای مقدار بعدی داشته باشد، می‌تواند iterator سفارشی باشد. وقتی مقدار دیگری نمانده است، `__next__()` باید StopIteration بالا بیندازد. برای پیاده‌سازی ساده‌تر، اغلب generator کافی است.

۹۳. **تفاوت خواندن فایل با read و readline چیست؟** `read()` مقدار مشخصی از داده یا کل محتوا را می‌خواند؛ `readline()` یک خط را در هر بار فراخوانی می‌دهد. برای فایل‌های بزرگ معمولاً پیمایش مستقیم: `for line in file:` از بارگذاری کل فایل در حافظه مناسب‌تر است.

۹۴. **حالت‌های رایج بازکردن فایل کدام‌اند؟** `r` خواندن، `w` نوشتن از ابتدا (و بازنویسی محتوا)، `a` افزودن به انتها، و `x` ساخت فایل تازه بدون بازنویسی فایل موجود است. افزودن `b` حالت باینری و افزودن `t` حالت متنی را مشخص می‌کند؛ بهتر است encoding متن، مانند UTF-8، صریح تعیین شود.

۹۵. **module docstring و package چه فایده‌ای دارند؟** docstring در ابتدای فایل، کلاس یا تابع، هدف و نحوه استفاده از آن را توضیح می‌دهد و از طریق `__doc__` یا ابزار `help` در دسترس است. تقسیم برنامه به ماژول‌ها و پکیج‌های مسئولیت‌محور، استفاده مجدد، آزمون و نگهداری را آسان‌تر می‌کند.

۹۶. **تفاوت کد هم‌زمان و async چیست؟** در اجرای هم‌زمان، هر عملیات معمولاً پیش از رفتن به بعدی تکمیل می‌شود. `asyncio` با `await` و `async def` امکان می‌دهد کارهای ورودی/خروجی در زمان انتظار به هم واگذار شوند؛ این روش به خودی خود محاسبات CPU-bound را سریع‌تر نمی‌کند.

۹۷. **GIL چیست؟** در پیاده‌سازی مرجع CPython، قفل سراسری مفسر (GIL) محدودیتی است که در یک فرایند معمولاً اجرای هم‌زمان bytecode پایتون را میان threadها کنترل می‌کند. threadها همچنان برای I/O مفیدند؛ برای کار CPU-bound می‌توان multiprocessing یا پیاده‌سازی‌ها و قابلیت‌های مناسب نسخه مورد استفاده را بررسی کرد.

۹۸. **چه روش‌هایی برای آزمون کد پایتون وجود دارد؟** آزمون واحد با چارچوب‌هایی مانند unittest یا ابزارهای بیرونی مانند pytest رفتار تابع‌ها و کلاس‌ها را بررسی می‌کند. آزمون‌ها باید حالت معمول، ورودی مرزی و خطاها را پوشش دهند؛ تست خودکار، تغییرات بعدی را مطمئن‌تر می‌کند.

۹۹. **چند اصل برای نوشتن کد خوانا در پایتون چیست؟** نام‌گذاری روشن، تابع‌های کوچک با مسئولیت مشخص، رعایت قراردادهای PEP 8، پرهیز از تکرار، مستندسازی تصمیم‌های غیرآشکار و استفاده از ابزارهای آزمون و قالب‌بندی از اصول مهم‌اند. کد خوانا هزینه دیباگ و همکاری تیمی را کاهش می‌دهد.

۱۰۰. **در مصاحبه فنی پایتون چگونه پاسخ تشریحی قوی بدهیم؟** ابتدا تعریف دقیق مفهوم را بیان کنید، سپس یک مثال کوچک یا مورد کاربردی بیاورید و در پایان محدودیت یا تفاوت مهم را توضیح دهید. اگر سؤال خروجی کد است، گام‌به‌گام رفتار را تحلیل کنید؛ اگر پاسخ را نمی‌دانید، فرض‌ها را شفاف بیان کنید و مسیر استدلال خود را نشان دهید.