1. What is the output of the following code? **C**

```
def calculate (num1, num2=4):
    res = num1 * num2
    print(res)

calculate(5, 6)
```

A. 20　B. The program executed with errors　C. 30

详细解答

这道题的考察 **默认参数**与**传入参数**的优先级问题。

- **参数定义**：在 `def calculate(num1, num2=4)` 中，`num2` 被赋予了默认值 4。
- **函数调用**：代码执行了 `calculate(5, 6)`。
- **覆盖机制**：虽然 `num2` 有默认值，但由于我们在调用时传入了第二个参数 `6`，这个新的值会**覆盖**掉原来的默认值 4。

计算步骤：

(a) 赋值：`num1 = 5`, `num2 = 6`

(b) 计算：$res = 5 \times 6$

(c) 输出：30

如果调用 `calculate(5)`，结果是 20。

2. What is the output of the following code: **A**

```
salary = 8000

def printSalary():
    salary = 12000
    print("Salary:", salary)

printSalary()
```

A. Salary: 12000　B. Salary: 8000　C. The program failed with errors

详细解答

这道题考察的是 **局部变量**与 **全局变量**的区别。

- **全局变量**：在函数外部定义的 `salary = 8000` 是全局变量，它在整个脚本范围内有效。
- **局部变量**：在函数 `printSalary()` 内部定义的 `salary = 12000` 是一个局部变量。
- **覆盖**：当函数内部存在与全局变量同名的变量时，函数会优先使用自己的局部变量。

执行流程：

(a) 程序首先在全局空间定义 `salary = 8000`。

(b) 调用 `printSalary()` 函数。

(c) 在函数内部，重新创建了一个局部变量 `salary` 并赋值为 `12000`。

(d) `print` 语句执行时，查找当前作用域，发现了局部变量 `12000`，因此将其打印。

结论：函数内部的修改不会影响外部全局变量的值（除非使用 `global` 关键字），但打印动作发生在内部，所以输出为 **Salary: 12000**。

3. What is the output of the following code? **B**

```
str = "Beijing University of Technolgy"
print (str[1:3])
```

A. Be　B. ei　C. Bei　D. eij

详细解答

这道题考察的是 Python 的 **切片**语法：`string[start:end]`。

- **索引从 0 开始**：Python 字符串的第一个字符索引是 0。
  - 索引 0: `'B'`
  - 索引 1: `'e'`
  - 索引 2: `'i'`

  - 索引 3: `'j'`

- **左闭右开原则**：切片操作包含 `start` 索引处的字符，但**不包含** `end` 索引处的字符。
- **计算区间**：`[1:3]` 表示从索引 1 开始，取到索引 3 之前（即只取索引 1 和 2）。

执行流程：

(a) 找到 `str[1]`，对应字母为 `'e'`。

(b) 找到 `str[2]`，对应字母为 `'i'`。

(c) 停止于 `str[3]` 之前，不包含 `'j'`。

结论：组合结果为 `"ei"`。

4. What is the output of the following code? **B**

```
p, q, r = 10, 20, 30
print(p, q, r)
```

A. 10 20　　B. 10 20 30　　C. Error: invalid syntax

详细解答

Python 的**多重赋值**功能。

- **语法逻辑**：Python 允许在一行代码中为多个变量赋值。等号左边的变量名与等号右边的值按**位置顺序**一一对应。
- **对应关系**：
  - `p` 对应第一个值 10
  - `q` 对应第二个值 20
  - `r` 对应第三个值 30
- **打印输出**：`print(p, q, r)` 会依次打印这三个变量，默认以空格分隔。

关键规则：等号左右两边的元素数量必须**严格相等**。

- 如果写成 `p, q = 10, 20, 30`，程序会抛出 `ValueError`（太多值无法解包）。
- 如果写成 `p, q, r = 10, 20`，程序会抛出 `ValueError`（值不够，无法解包）。

结论：变量与数值数量匹配，赋值成功。输出为 **10 20 30**。

5. What is the output of the following code? **A**

```
x = 36 / 4 * (3 + 2) * 4 + 2
print(x)
```

A. 182.0　B. 37　C. 117　D. The program executed with errors

详细解答

运算符优先级和 Python 3 的除法机制。

- **优先级顺序：**括号 () → 乘除 (* /) → 加法 (+)。
- **同级运算：**乘法和除法处于同一优先级，按照**从左到右**的顺序执行。
- **浮点数结果：**在 Python 3 中，使用单个斜杠 `/` 进行除法运算，其结果总是 **float (浮点数)**，即使能整除也不例外。

逐步计算过程：

(a) **处理括号：**`(3 + 2) = 5`

(b) **从左到右计算乘除：**

- `36 / 4 = 9.0`（注意：此处产生浮点数）
- `9.0 * 5 = 45.0`
- `45.0 * 4 = 180.0`

(c) **最后计算加法：**`180.0 + 2 = 182.0`

结论：因为除法操作的存在，结果必须带有小数点。故正确答案是 **182.0**。

6. What is the output of the following code? **A**

```
for i in range(10, 15, 1):
```

```
    print(i, end=', ')
```

A. 10, 11, 12, 13, 14, B. 10, 11, 12, 13, 14, 15

详细解答

Python 的 `range()` 函数参数以及 `print()` 函数的 `end` 参数。

- **range(start, stop, step)：**
  - `start = 10`：计数从 10 开始（包含）。
  - `stop = 15`：计数到 15 结束（**不包含**）。这是 Python 典型的“左闭右开”原则。
  - `step = 1`：步长为 1，即每次递增 1。
- **print(..., end=', ')：**
  - 默认情况下，`print` 会在输出后换行。
  - 使用 `end=', '` 后，每次打印结束后不再换行，而是追加一个逗号和一个空格。

执行流程：

(a) 第一轮：$i = 10$，打印 `10,`

(b) 第二轮：$i = 11$，打印 `11,`

(c) 第三轮：$i = 12$，打印 `12,`

(d) 第四轮：$i = 13$，打印 `13,`

(e) 第五轮：$i = 14$，打印 `14,`

(f) 达到 `stop` 值 15，循环结束。

结论：循环生成的序列是 $10, 11, 12, 13, 14$，末尾均带有指定的格式。

7. What is the output of the following code? **A**

```
listOne = [20, 40, 60, 80]
listTwo = [20, 40, 60, 80]

print(listOne == listTwo)
```

A. True　B. False

详细解答

相等运算符 (Equality Operator) == 的工作原理。

- **内容比较 (==)**：在 Python 中，== 运算符用于比较两个对象的**值（内容）**是否相等。对于列表而言，如果两个列表包含相同顺序、相同数量且值相等的元素，则结果为 `True`。
- **对象标识 (is)**：需要注意的是，如果是使用 `listOne is listTwo` 比较，结果将是 `False`，因为 `is` 比较的是它们是否为内存中的同一个对象（即 ID 是否相同）。

代码分析：

(a) `listOne` 的值为 `[20, 40, 60, 80]`。

(b) `listTwo` 的值为 `[20, 40, 60, 80]`。

(c) 两个列表的元素及其排列顺序完全一致。

结论：由于 `listOne` 和 `listTwo` 的内容完全相同，表达式 `listOne == listTwo` 返回 **True**。

8. What is the output of the following code? **B**

```
listOne = [20, 40, 60, 80]
listTwo = [20, 40, 60, 80]

print(listOne is listTwo)
```

A. True　B. False

详细解答

这道题考察的是 Python 中的 **身份运算符 (Identity Operator)** `is` 与相等运算符 == 的本质区别。

- **== (Equality)**：比较的是两个对象的**值**是否相等（Content）。
- **is (Identity)**：比较的是两个对象是否指向**内存中的同一个地址**（Object Identity/Memory Address）。

代码分析：

(a) `listOne = [20, 40, 60, 80]`：Python 在内存中创建了一个新的列表对象。

(b) `listTwo = [20, 40, 60, 80]`：Python 又创建了一个新的列表对象。虽然内容一样，但它们是两个独立的容器。

(c) `listOne is listTwo`：系统会检查 `id(listOne)` 是否等于 `id(listTwo)`。

核心结论：尽管两个列表的内容完全一致，但它们存储在内存的不同位置。由于它们是两个不同的对象，`is` 的结果为 **False**。

延伸思考：如果执行 `listThree = listOne`，然后再比较 `listOne is listThree`，结果才会是 `True`，因为此时两个变量指向了同一个内存地址。

9. What is the output of the following code? **A**

```
var = "James" * 2 * 3
print(var)
```

A. JamesJamesJamesJamesJamesJames
B. JamesJamesJamesJamesJames
C. Error: invalid syntax

详细解答

这道题考察的是 Python 中的 **字符串乘法 (String Replication)** 运算符。

- **乘法运算符 `*`：**当乘法运算符用于字符串和整数之间时，它表示将该字符串重复指定的次数。
- **结合性：**乘法运算在 Python 中具有**左结合性**，即从左向右依次计算。

执行流程分析：

(a) **第一步：**计算 `"James" * 2`。

- 结果为 `"JamesJames"` (重复 2 次)。

(b) **第二步：**计算 `"JamesJames" * 3`。

- 将上一步得到的字符串再重复 3 次。
- 相当于总共重复了 $2 \times 3 = 6$ 次。

结论：最终变量 `var` 存储的是连续 6 个"James"。`"James" + "James" + "James" + "James" + "James" + "James"` 故正确答案是 **A**。

注意：字符串只能与**整数**相乘。如果尝试 `"James" * "Bond"`，程序会抛出 `TypeError`。

10. What is the output of the following code? **C**

```
valueOne = 5 ** 2
print(valueOne)
```

A. Error: invalid syntax　B. 15　C. 25　D. 125

详细解答

这道题考察的是 Python 中的 **幂运算符 (Exponentiation Operator)**。

- **算术运算符 `**`：**在 Python 中，双星号 `**` 代表幂运算，即计算左操作数的右操作数次幂（x^y）。
- **区别于其他语言：**在某些编程语言（如 C++ 或 Java）中，计算方幂通常需要调用 `pow()` 函数或使用`^`（注意：在 Python 中`^` 是位运算符异或 XOR）。

计算过程：

(a) 表达式为 `5 ** 2`。

(b) 对应数学公式：$5^2 = 5 \times 5$。

(c) 计算结果为 25。

知识拓展：

- **浮点数幂运算：**如果底数或指数是浮点数，结果也会是浮点数。例如 `5 ** 2.0` 结果为 `25.0`。
- **开方运算：**你也可以使用 `**` 进行开方，例如 `25 ** 0.5` 结果为 `5.0`。

结论：计算结果为整型数字 25。

11. What is the output of the following code? **D**

```
var1 = 1
var2 = 2
var3 = "3"

print(var1 + var2 + var3)
```

A. 6　B. 33　C. 123　D. Error. Mixing operators between numbers and strings are not supported

详细解答

这道题考察的是 Python 的 **强类型特性 (Strong Typing)** 以及加法运算符 + 在不同数据类型间的行为。

- **数据类型识别：**
  - `var1` 和 `var2` 是 **整型 (int)**。
  - `var3` 是 **字符串 (str)**，因为它被包裹在引号中。
- **运算顺序：**
  (a) 首先执行 `var1 + var2`，即 $1+2$，结果为整型 3。
  (b) 接着尝试执行 `3 + var3`，即 `3 + "3"`。

为什么会报错？ Python 不允许直接将 **整型 (int)** 和 **字符串 (str)** 使用 + 号相加：

- 对于数字，+ 代表算术加法。
- 对于字符串，+ 代表字符串拼接（Concatenation）。
- Python 无法自动确定你是想把字符串转成数字做加法，还是把数字转成字符串做拼接。

报错类型：程序会抛出 `TypeError: unsupported operand type(s) for +: 'int' and 'str'`。
结论：混合使用数字和字符串进行加法运算而不进行显式类型转换是不受支持的。故正确答案是 **D**。

修正方法：

- 若要结果为 6：使用 `var1 + var2 + int(var3)`。
- 若要结果为 `"33"`：使用 `str(var1 + var2) + var3`。

12. A string literal in Python must be enclosed in _____. **d**

a. parentheses
b. single-quotes
c. double-quotes
d. either single-quotes or double-quotes

详细解答

这道题考察的是 Python 中 **字符串字面量 (String Literals)** 的定义规则。

- **引号的使用**：Python 在设计上非常灵活，允许使用单引号 (`'...'`) 或双引号 (`"..."`) 来创建字符串。
- **功能等价**：在 Python 中，`'Hello'` 和 `"Hello"` 是完全等价的，生成的字符串对象没有任何区别。
- **嵌套规则**：
  - 如果字符串内部包含单引号，可以使用双引号包裹，如 `"It's Python"`。
  - 如果字符串内部包含双引号，可以使用单引号包裹，如 `'He said "Hello"'`。

选项分析：

a. **parentheses (圆括号)**：通常用于元组定义、函数调用或控制运算优先级，不能用于定义字符串。

b. **single-quotes (单引号)**：可以定义字符串，但不是唯一方式。

c. **double-quotes (双引号)**：可以定义字符串，但也不是唯一方式。

d. **either single-quotes or double-quotes**：准确描述了 Python 的语法规则。

结论：Python 字符串字面量必须由成对的单引号或双引号包围。故正确答案是 **d**。

知识拓展：除了单双引号，Python 还支持**三引号** (`'''...'''` 或 `"""..."""`)，用于定义多行字符串或编写文档字符串 (docstrings)。

13. This symbol marks the beginning of a comment in Python. **d**

a. `&`　b. `*`　c. `**`　d. `#`

详细解答

这道题考察的是 Python 中的 **注释 (Comments)** 语法。

- **注释的作用**：注释是写给开发者看的文字，用于解释代码逻辑。Python 解释器在执行代码时会完全忽略掉注释内容。
- **单行注释符号**：在 Python 中，单行注释以 `#` 号开头。从 `#` 开始直到该行结束的所有内容都被视为注释。

选项分析：

a. `&`：位运算符（AND），不用于注释。

b. `*`：算术乘法运算符或解包运算符，不用于注释。

c. `**`：幂运算符（如 5^2），不用于注释。

d. `#`：**正确**。这是 Python 定义单行注释的标准符号。

代码示例：

```
# 这是一个注释，解释下面的变量
radius = 5  # 计算圆的半径
```

结论：在 Python 中，`#` 符号标识注释的开始。故正确答案是 **d**。

对比其他语言：

- C/C++/Java 使用 `//` 。
- HTML/XML 使用 。
- SQL 使用 `--` 。

14. Which of the following statements will cause an error? **b**

a. `x = 17`

b. `17 = x`

c. `x = 99999`

d. `x = "17"`

详细解答

这道题考察的是 Python 中的 **赋值运算符 (=)** 的语法规则以及 **左值 (L-value)** 的概念。

- **赋值逻辑：**赋值运算符 `=` 的作用是将等号**右侧**的值（表达式的结果）存储到等号**左侧**的变量空间中。
- **左值要求：**等号左侧必须是一个可以存储数据的“容器”，通常是一个变量名。它不能是一个不可改变的字面量（Literal），如数字或字符串。

选项分析：

a. `x = 17`：合法。将整数 17 赋值给变量 `x`。

b. `17 = x`：**非法 (Error)**。数字 17 是一个常数（字面量），它不是一个变量，无法被赋值。这会触发 `SyntaxError: cannot assign to literal`。

c. `x = 99999`：合法。Python 的整数支持任意精度，这是一个非常大的整数赋值。

d. `x = "17"`：合法。将字符串 `"17"` 赋值给变量 `x`。

结论：赋值语句的方向性是固定的：变量 `=` 值。尝试给字面量赋值会导致语法错误。故正确答案是 **b**。

记忆要点：在编程中，可以把变量想象成“盒子”，把值想象成“物品”。可以把物品放进盒子里（`box = item`），但不能把盒子放进物品里（`item = box`）。

15. This operator performs integer division. **a**

a. `//`
b. `%`
c. `**`
d. `/`

详细解答

这道题考察的是 Python 中的 **地板除 (Floor Division)**，也就是整数除法运算符。

- **运算符 `//`**：执行除法运算并向下取整到最接近的整数（即丢弃小数部分）。
- **与普通除法的区别**：
  - `7 / 2` 结果为 `3.5`（浮点数除法）。
  - `7 // 2` 结果为 `3`（整数除法）。

选项分析：

a. `//`：**正确**。用于执行整数除法。

b. `%`：取模运算符，返回除法的**余数**（例如 `7 % 2` 结果为 `1`）。

c. `**`：幂运算符（例如 `5 ** 2` 结果为 `25`）。

d. `/`：普通除法运算符，在 Python 3 中总是返回浮点数。

结论：若要执行整数除法并获取商的整数部分，必须使用 `//`。故正确答案是 **a**。

小贴士：如果操作数中包含浮点数（如 `7.0 // 2`），结果虽然仍是整数部分，但类型会是浮点型（即 `3.0`）。

16. This is an operator that raises a number to a power. **c**

a. `%`
b. `*`
c. `**`
d. `/`

详细解答

这道题考察的是 Python 中的 **幂运算符 (Exponentiation Operator)**，用于计算一个数的 n 次方。

- **运算符 `**`**：在 Python 中，双星号用于表示底数的指数幂运算。例如，a^b 在代码中写作 `a ** b`。

- **运算顺序：**幂运算符在算术运算符中拥有非常高的优先级（高于乘、除、加、减）。

选项分析：

a. `%`：取模运算符（Modulo），用于返回除法的余数。

b. `*`：乘法运算符。

c. `**`：**正确**。幂运算符。

d. `/`：除法运算符（结果为浮点数）。

代码示例：

```
print(2 ** 3)  # 结果为 8 (2 * 2 * 2)
print(10 ** 2) # 结果为 100
```

结论：用于执行次方运算的运算符是 `**`。故正确答案是 **c**。

易错点：在很多其他语言中（如 Excel 或某些计算器语法），幂运算使用 ^ 符号。但在 Python 中，^ 是按位异或运算符（Bitwise XOR），千万不要混淆。

17. This operator performs division, but instead of returning the quotient it returns the remainder. **a**

a. `%`

b. `*`

c. `**`

d. `/`

详细解答

这道题考察的是 Python 中的 **取模运算符 (Modulus Operator)**。

- **运算符 `%`：**执行除法运算，但返回的是除法后的**余数** (Remainder)，而不是商 (Quotient)。
- **数学表示：**如果执行 $a \div b = q \ldots r$，那么 `a % b` 的结果就是 r。

选项分析：

a. `%`：**正确**。取模运算符，返回余数。

b. `*`：乘法运算符。

c. `**`：幂运算符。

d. `/`：除法运算符（返回浮点数商）。

实际应用示例：

- **判断奇偶性：** `n % 2` 如果等于 0 则是偶数，等于 1 则是奇数。
- **限制范围：** 例如 `index % length` 可以确保索引永远在数组长度范围内循环。

示例代码：

```
print(10 % 3)  # 结果为 1 (10 / 3 = 3 余 1)
print(12 % 4)  # 结果为 0 (12 能被 4 整除)
```

结论：返回除法余数的运算符是 `%`。故正确答案是 **a**。

18. Suppose the following statement is in a program:

```
price = 99.0
```

After this statement executes, the `price` variable will reference a value of which data type? **b**

a. int
b. float
c. currency
d. str

详细解答

这道题考察的是 Python 中的 **数据类型识别**，特别是如何区分整型和浮点型。

- **浮点型 (float)：**在 Python 中，任何包含**小数点**的数字字面量都会被识别为 `float` 类型。即使小数点后面是零（如 `99.0`），它依然被视为浮点数。
- **动态类型：**Python 是一种动态类型语言，变量的类型由它所引用的对象决定。

选项分析：

a. `int` (整型)：用于表示没有小数部分的整数（如 `99`）。

b. `float` (浮点型)：**正确**。用于表示实数，包括带小数点的数字。

c. `currency`：**错误**。Python 原生数据类型中并没有名为 `currency` 的类型（虽然可以使用 `decimal` 模块处理货币）。

d. `str` (字符串)：用于表示文本，必须用引号包裹（如 `"99.0"`）。

验证方法：在 Python 中，你可以使用 `type()` 函数来检查变量的类型：

```
price = 99.0
print(type(price))  # 输出: <class 'float'>
```

结论：由于数字中包含小数点，该变量引用的数据类型是 `float`。故正确答案是 **b**。

19. Which built-in function can be used to read input that has been typed on the keyboard?

a. input()
b. get_input()
c. read_input()
d. keyboard()

详细解答

这道题考察的是 Python 中最基本的 **标准输入 (Standard Input)** 函数。

- **input() 函数：**这是 Python 的内置函数，用于暂停程序的执行，等待用户从键盘输入文本。
- **工作机制：**
  - 程序运行到 `input()` 时会阻塞，直到用户按下回车键（Enter）。
  - 该函数总是将用户的输入作为 **字符串 (string)** 返回，即使输入的是数字。
  - 可以接受一个可选的参数作为“提示信息”（Prompt）。

选项分析：

a. `input()`：**正确**。这是官方定义的唯一用于从键盘读取输入的内置函数。

b. `get_input()`：错误。Python 中没有这个内置函数（某些库可能有类似命名的函数，但不是标准内置）。

c. `read_input()`：错误。这不是标准内置函数。

d. `keyboard()`：错误。这不是函数，通常在底层驱动或第三方库中会提到键盘，但不是用来读取字符串输入的函数。

代码示例：

```
name = input("Enter your name: ")
print("Hello, " + name)
```

结论：用于读取键盘输入的内置函数是 `input()`。故正确答案是 **a**。

提醒：在 Python 2 中，对应的函数是 `raw_input()`，而 `input()` 会尝试评估输入的内容。但在 **Python 3** 中，我们统一使用 `input()`。

20. Which built-in function can be used to convert an int value to a float? **b**

a. `int_to_float()`
b. `float()`
c. `convert()`
d. `int()`

详细解答

这道题考察的是 Python 中的 **显式类型转换 (Explicit Type Conversion)**，也称为“强制类型转换”。

- **float() 函数：**这是 Python 的内置构造函数，用于将其他数据类型（如整数或符合数字格式的字符串）转换为 **浮点数 (float)**。
- **转换逻辑：**当一个整数被转换为浮点数时，Python 会在数值后添加小数点。例如，整数 10 会变成浮点数 10.0。

选项分析：

a. `int_to_float()`：错误。Python 并没有这种命名的内置函数。

b. `float()`：**正确**。这是标准的转换函数。

c. `convert()`：错误。这不是 Python 的内置函数。

d. `int()`：错误。这个函数的作用正好相反，它是将其他类型转换为**整数**（通过截断小数部分）。

代码示例：

```
number = 5
result = float(number)
print(result)         # 输出：5.0
print(type(result))   # 输出：<class 'float'>
```

结论：用于将整数转换为浮点数的内置函数是 `float()`。故正确答案是 **b**。

知识拓展：这种转换在进行除法运算或需要更高精度计算时非常有用。即使是字符串 `"3.14"`，也可以通过 `float("3.14")` 转换为数值类型。

21. Choosing the **wrong** way to create a string literal `Ault'Kelly`: **A**

A. `str1 = 'Ault\\'Kelly'`

B. `str1 = 'Ault\'Kelly'`

C. `str1 = """Ault'Kelly"""`

详细解答

这道题考察的是 Python 字符串中的 **转义字符 (Escape Characters)** 以及如何处理字符串内部的引号冲突。

- **核心规则：**如果字符串使用单引号 `' '` 定义，那么内部出现的单引号必须使用反斜杠 `\` 进行转义（即 `\'`），否则 Python 会认为字符串提前结束。
- **目标字符串：**我们需要创建的内容是 `Ault'Kelly`（中间有一个单引号，没有反斜杠）。

选项分析：

A. `'Ault\\'Kelly'`：**错误的做法 (符合题意)**。

- `\\` 会被解析为一个字面意义上的反斜杠。
- 紧随其后的 `'Kelly'` 中的单引号由于没有被转义，会直接闭合前面的单引号，导致语法错误（SyntaxError）。
- 即使不报错，它的本意也是试图创建一个带有反斜杠的字符串，不符合目标。

B. `'Ault\'Kelly'`：**正确的做法**。反斜杠转义了单引号，使其成为字符串内容的一部分。

C. `"""Ault'Kelly"""`：**正确的做法**。使用三引号定义字符串时，内部可以直接包含单引号或双引号而无需转义。

结论：选项 A 的写法会导致语法错误或逻辑错误，是创建该特定字符串的“错误方式”。故正确答案是 **A**。

小技巧：如果你不想处理复杂的转义，最简单的方法是交叉使用引号。例如，使用双引号包裹含有单引号的字符串：`"Ault'Kelly"`。

22. What is the output of the following code? **B**

```
x = 50
def fun1():
    x = 25
    print(x)

fun1()
```

A. NameError　B. 25　C. 50

详细解答

这道题考察的是 Python 的 **变量作用域 (Variable Scope)**，特别是 **全局变量**与 **局部变量**的区别。

- **全局变量 (Global Variable)**：在函数外部定义的变量（如第一行的 `x = 50`）。它在整个程序范围内都是可见的。
- **局部变量 (Local Variable)**：在函数内部定义的变量（如函数内的 `x = 25`）。它的生命周期仅限于函数运行期间，且会“遮蔽”（Shadow）

同名的全局变量。

代码执行流程分析：

(a) `x = 50`：在全局命名空间创建变量 `x`。

(b) `def fun1():`：定义函数，此时不执行函数体。

(c) `fun1()`：调用函数。

- 进入函数内部，执行 `x = 25`。这会在 `fun1` 的局部命名空间创建一个新的 `x`。
- 执行 `print(x)`：Python 会遵循 **LEGB 规则**（Local → Enclosing → Global → Built-in）寻找变量。由于在 Local（局部）作用域找到了 `x`，所以直接打印局部变量的值。

结论：函数内部的 `print` 输出的是局部变量的值 25。即使外部有一个 `x = 50`，在函数内部它也被局部定义的 `x` 覆盖了。故正确答案是 **B**。

思考：如果在调用 `fun1()` 之后，在函数外部再执行一次 `print(x)`，输出结果会是多少？答案是 **50**，因为函数内部的局部赋值不会影响外部的全局变量（除非显式使用 `global` 关键字）。

23. What is the output of the following code? **A**

```
M = 50
def fun1():
    m = 25

fun1()
print(m)
```

A. NameError　B. 25　C. 50

详细解答

这道题考察的是 Python 的 ** 大小写敏感性 (Case Sensitivity)** 以及 ** 变量作用域 (Scope)**。

- **大小写敏感：**Python 严格区分变量名的大小写。在这个例子中，大写的 `M` 和小写的 `m` 是两个完全不同的变量名。

- **变量生命周期：**
  - `M = 50`：定义了一个全局变量 **大写 M**。
  - `m = 25`：在函数内部定义了一个局部变量 **小写 m**。

执行流程分析：

(a) `M = 50`：程序创建了全局变量 `M`。

(b) `fun1()` 被调用：

- 在函数作用域内创建了局部变量 `m`。
- 函数执行完毕，局部作用域销毁，局部变量 `m` 也随之消失。

(c) `print(m)`：程序在全局作用域查找小写的 `m`。

- **找不到变量**：全局作用域里只有 `M`，没有 `m`。

结论：由于在全局作用域中从未定义过小写的 `m`，试图打印它会导致程序崩溃。报错信息为：`NameError: name 'm' is not defined`。故正确答案是 **A**。

对比点：如果最后一行是 `print(M)`（大写），结果将是 `50`。如果函数内写的是 `global m; m = 25`，结果将是 `25`。

24. What is the result of `print(type([]) is list)`? **B**

A. False B. True

详细解答

这道题考察的是 Python 中 **类型检查 (Type Checking)** 的底层机制以及 `type()` 函数的返回值。

- **表达式拆解：**
  - `[]`：这是一个空的 **列表 (list)** 字面量。
  - `type([])`：调用内置函数 `type()`。对于列表对象，它返回的是列表类型对象，即 `<class 'list'>`。
  - `list`：这是 Python 的内置关键字/类名，代表列表类型。
- **身份比较 (`is`)：**

– `is` 运算符检查两个对象是否为同一个实例。在 Python 中，内置类型的类对象（如 `list`, `int`, `str`）在内存中是单例的。

逻辑推导：

(a) `type([])` 的结果精确等同于内置的 `list` 类。

(b) 因此，比较 `type([]) is list` 实际上是在问："列表实例的类型是否就是 `list` 类本身？"

(c) 答案显然是肯定的。

结论：该表达式评估为 **True**。故正确答案是 **B**。

知识拓展：虽然 `type(obj) is class` 可以工作，但在 Python 编程实践中，通常推荐使用 `isinstance([], list)` 来进行类型检查，因为 `isinstance` 还能处理子类继承的情况。

25. Which of the following cannot create a valid String? **D**

A. `str1 = 'str1'`

B. `str1 = "str1"`

C. `str1 = """str"""`

D. `str1 = str(Jessa)`

详细解答

这道题考察的是 Python 字符串的创建方式以及 **变量定义**与 **字面量**之间的区别。

- **字符串字面量：**必须被包裹在引号内（单引号、双引号或三引号）。
- **变量引用：**如果不带引号直接写一个单词（如 `Jessa`），Python 会将其视为一个变量名，并尝试在内存中寻找该变量对应的值。

选项分析：

A. `'str1'`：合法。使用单引号定义字符串。

B. `"str1"`：合法。使用双引号定义字符串。

C. `"""str"""`：合法。使用三引号定义多行字符串。

D. `str(Jessa)`：**非法 (Error)**。

- 这里 `Jessa` 没有加引号，Python 会认为它是一个名为 `Jessa` 的**变量**。
- 如果代码之前没有定义过 `Jessa = "some value"`，程序会抛出 `NameError: name 'Jessa' is not defined`。
- 即使定义了，这也是在做类型转换，而不是直接通过字面量创建字符串。

结论：选项 D 试图将一个未定义的变量传递给 `str()` 函数，这在标准语法中无法直接创建一个有效的字符串字面量。故正确答案是 **D**。

修正建议：如果你想创建内容为"Jessa" 的字符串，应该写成 `str1 = "Jessa"` 或者 `str1 = str("Jessa")`。

26. What is the data type of `print(type(10))`? **C**

A. float　B. integer　C. int

详细解答

这道题考察的是 Python 的 **内置数据类型名称**以及 `type()` 函数的返回结果。

- **数值识别**：数字 `10` 是一个不带小数点的数字，在 Python 中被归类为**整数**。
- **关键字名称**：在 Python 编程语言中，表示整数类型的官方关键字是 `int`。

选项辨析：

A. **float**：浮点型，用于表示带小数的数字（如 `10.0`）。

B. **integer**：这是数学上的称呼“整数”。虽然逻辑上正确，但它不是 Python 中的**编程关键字**。

C. **int**：**正确**。这是 Python 内部定义的类名。当你运行 `type(10)` 时，Python 会返回 `<class 'int'>`。

实验验证：

```
result = type(10)
print(result)
# 输出结果为: <class 'int'>
```

结论：根据编程语境，10 是 int 类型。

27. What is the data type of the following? **C**

```
aTuple = (1, 'Jhon', 1+3j)
print(type(aTuple[2:3]))
```

A. list　B. complex　C. tuple

详细解答

这道题考察的是 Python 中 **切片操作 (Slicing)** 的一个核心原则：** 切片操作返回的对象类型通常与原对象类型一致 **。

- **对象识别：**aTuple 是一个元组（Tuple），由圆括号定义。
- **索引 vs 切片：**
  - **索引 (Indexing)：**aTuple[2] 访问的是索引为 2 的具体元素。结果是 1+3j，其类型是 **complex**（复数）。
  - **切片 (Slicing)：**aTuple[2:3] 访问的是从索引 2 到 3（不包含 3）的 ** 范围 **。

关键知识点：在 Python 中，当你对一个序列（如 list, tuple, string）进行切片时，返回的永远是该序列的一个“子集”，其 ** 数据结构保持不变 **。

- 列表的切片结果依然是列表。
- 字符串的切片结果依然是字符串。
- **元组的切片结果依然是元组。**

计算过程：

(a) aTuple[2:3] 截取了从位置 2 开始到位置 3 结束的部分。

(b) 结果是一个只包含一个元素的元组：((1+3j),)。

(c) type() 函数识别该结果，返回 <class 'tuple'>。

结论：切片操作保留了容器类型。故正确答案是 **C**。

28. What is the output of the following code? **D**

```
print(bool(0), bool(3.14159), bool(-3), bool(1.0+1j))
```

A. False True False True
B. True True False True
C. True True False True
D. False True True True

详细解答

这道题考察的是 Python 中的 **布尔测试 (Truth Value Testing)**，即如何将其他数据类型转换为布尔值 (`True` 或 `False`)。

- **bool() 函数**：用于将一个值转换成布尔类型。
- **虚值 (Falsy Values)**：在 Python 中，以下数值会被评估为 `False`：
  - 定义为 False 的常量：`None`, `False`。
  - 任何数值类型的零：`0`, `0.0`, `0j`, `Decimal(0)`, `Fraction(0, 1)`。
  - 空的序列和集合：`''`, `()`, `[]`, `{}`, `set()`, `range(0)`。
- **实值 (Truthy Values)**：除了上述虚值外，绝大多数对象都被评估为 `True`。

逐项分析：

(a) `bool(0)`：整数零，结果为 **False**。

(b) `bool(3.14159)`：非零浮点数，结果为 **True**。

(c) `bool(-3)`：非零整数（注意：负数也是非零的），结果为 **True**。

(d) `bool(1.0+1j)`：非零复数，结果为 **True**。

结论：对应的布尔序列为 `False True True True`。故正确答案是 **D**。

核心总结：在数值转换中，** 只有 0 是 False**，其他任何数字（无论正负、小数还是复数）都是 True。

29. In Python 3, what is the output of `type(range(5))`? (What data type it will return.) **C**

A. int　B. list　C. range　D. None

详细解答

这道题考察的是 Python 3 中 **range 对象**的本质及其与 Python 2 的重要区别。

- **range() 函数**：在 Python 3 中，range() 不再返回一个列表，而是返回一个专门的 **可迭代对象 (Iterable)**，其类型名称就叫做 range。
- **惰性求值 (Lazy Evaluation)**：range 对象在内存中非常高效。无论你定义 range(5) 还是 range(1000000)，它在内存中占用的空间是固定的，因为它只存储起始值、终止值和步长，只有在循环迭代时才产生具体的数字。

选项分析：

A. int：错误。range 是一个序列容器，不是单个整数。

B. list：错误。这是 Python 2 的行为。在 Python 3 中，如果你需要列表，必须显式转换：list(range(5))。

C. range：**正确**。Python 3 为其定义了专属的类。

D. None：错误。函数有明确的返回值。

实验验证：

```
r = range(5)
print(type(r))
# 输出结果为: <class 'range'>
```

结论：在 Python 3 中，range(5) 返回的是一个类型为 range 的对象。故正确答案是 **C**。

30. What is the output of the following code? **B**

```
def func1():
    x = 50
    return x
func1()
print(x)
```

A. 50　B. NameError　C. None　D. 0

详细解答

这道题再次深入考察了 **局部作用域 (Local Scope)** 的生命周期以及变量的访问权限。

- **局部变量的定义：**在函数 `func1()` 内部定义的变量 `x` 是一个局部变量。它仅存在于该函数的命名空间内。
- **作用域的隔离：**当函数执行完毕并返回结果后，该函数的作用域就会被销毁。在函数外部（全局作用域）是无法直接访问函数内部定义的局部变量的。

代码执行流程分析：

(a) **定义函数：**定义 `func1`，其中包含局部变量 `x`。

(b) **调用函数 `func1()`：**函数被执行，返回了数值 `50`。但请注意，程序 ** 并没有将这个返回值赋值给任何全局变量 **（例如没有写成 `result = func1()`）。

(c) **执行 `print(x)`：**此时程序尝试在全局作用域寻找变量 `x`。

- 全局作用域中没有定义过 `x`。
- 函数内部的 `x` 已经随着函数结束而不可见。

结论：由于在执行 `print(x)` 的位置找不到变量 `x` 的定义，Python 解释器会抛出 **NameError: name 'x' is not defined**。故正确答案是 **B**。

如何修正代码以输出 50？如果你希望打印出那个值，必须接收函数的返回值：

```
x = func1()   # 将函数的返回值赋给全局变量 x
print(x)      # 此时输出 50
```

31. What is the output of `print(10 - 4 * 2)`? **A**

A. 2　B. 12

详细解答

这道题考察的是 Python 中的 **运算符优先级 (Operator Precedence)**。

- **优先级规则：**在 Python 的算术运算中，乘法 (`*`)、除法 (`/`)、取模 (`%`) 和整除 (`//`) 的优先级高于加法 (`+`) 和减法 (`-`)。这与标准数学中的“先乘除后加减”原则一致。
- **计算步骤：**
  (a) 首先计算乘法部分：`4 * 2 = 8`。
  (b) 然后执行减法运算：`10 - 8 = 2`。

选项分析：

A. **2：正确**。按照先乘法后减法的顺序计算得出。

B. **12：**错误。这是在错误地先计算减法（$10-4=6$）再计算乘法（$6\times 2=12$）时得到的结果。

知识拓展：如果你希望先执行减法，必须使用**括号**来显式改变优先级：`(10 - 4) * 2` 的结果才是 `12`。

结论：由于乘法优先级更高，计算结果为 **2**。故正确答案是 **A**。

32. What is the output of `print(2 ** 3 ** 2)`? **B**

A. 64　B. 512

详细解答

这道题考察的是 Python 运算符中一个非常特殊且容易被忽视的规则：**幂运算符 (`**`) 的 结合性 (Associativity)**。

- **结合性定义：**当一个表达式中出现多个优先级相同的运算符时，结合性决定了计算的方向。
- **右结合性 (Right-to-Left)：**绝大多数 Python 算术运算符（如 `+`, `-`, `*`, `/`）都是**左结合**的。但 **幂运算符 (`**`) 是右结合的**。

计算过程详解：对于表达式 `2 ** 3 ** 2`：

(a) **第一步：**由于是右结合，先计算右侧的指数部分：$3^2=9$。

(b) **第二步：**将得到的结果作为左侧底数的指数：2^9。

(c) **第三步：**计算 $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 512$。

选项分析：

A. **64：**错误。这是按照左结合方向计算的结果：$(2^3)^2 = 8^2 = 64$。

B. **512：正确**。按照 Python 官方定义的右结合方向计算得出。

结论：在 Python 中，`a ** b ** c` 等价于 `a ** (b ** c)`。故正确答案是 **B**。

33. What is the value of the following Python expression? **A**

```
print(36 / 4)
```

A. 9.0　B. 9

详细解答

这道题考察的是 Python 3 中 ** 除法运算符 (Division Operator)** 的返回类型特性。

- **真除法 (True Division)：**在 Python 3 中，使用单个斜杠 `/` 执行的是真除法。
- **类型规则：**无论操作数（被除数和除数）是整数还是浮点数，`/` 运算的结果“永远是一个浮点数 (float)”。即使结果是一个完美的整数，Python 也会自动在后面补上 `.0`。

选项分析：

A. **9.0：正确**。这是 Python 3 的标准行为。

B. **9：**错误。在 Python 2 中，两个整数相除会得到整数。但在 Python 3 中，必须使用双斜杠 `//`（整除运算符）才会得到整数 `9`。

代码实验对比：

```
print(36 / 4)    # 输出: 9.0 (float)
print(36 // 4)   # 输出: 9 (int)
```

结论与答案修正：根据 Python 3 的语言规范，结果应当带有小数点。

34. What is the output of `print(2 * 3 ** 3 * 4)`? **A**

A. 216　B. 864

详细解答

这道题考察的是 Python 中 **混合运算符的优先级 (Operator Precedence)**，重点在于乘法 (*) 与幂运算 (**) 的先后顺序。

- **优先级排序：**在 Python 算术运算中，幂运算符 ** 的优先级高于乘法 *、除法 / 和取模 %。
- **计算方向：**当优先级相同时（如表达式中有两个乘法），按照从左到右的顺序执行。

逐步计算过程：表达式：`2 * 3 ** 3 * 4`

(a) **第一步（最高优先级）：**计算幂运算 3^3。

$$3 \times 3 \times 3 = 27$$

此时表达式变为：`2 * 27 * 4`

(b) **第二步（从左向右乘）：**计算 2×27。

$$2 \times 27 = 54$$

此时表达式变为：`54 * 4`

(c) **第三步：**计算最后一步乘法。

$$54 \times 4 = 216$$

选项分析：

A. **216：正确**。严格遵循先幂运算后乘法的原则。

B. **864：**错误。这是在错误地先计算了前面的乘法（$2 \times 3 = 6$），然后计算 $6^3 \times 4$ 得到的结果。

结论：幂运算永远优先于乘法。故正确答案是 **A**。

35. What is the output of `print(2 % 6)`? **C**

A. ValueError　B. 0.33　C. 2

详细解答

这道题考察的是 Python 中的 **取模运算符 (Modulus Operator)**，符号为 %。

- **运算符定义：**取模运算返回两个数相除后的 **余数 (Remainder)**。
- **基本公式：**对于 $a\%b$，其计算逻辑是找到一个整数 q，使得 $a = b \times q + r$，其中 r 就是余数（且 $|r| < |b|$）。

逻辑推导：当我们计算 `2 % 6` 时：

(a) **做除法：**$2 \div 6$。

(b) **求商：**因为 2 比 6 小，所以商为 **0**。

(c) **求余数：**$2 - (6 \times 0) = 2$。

关键规律：在取模运算中，如果 ** 左侧的数字（被除数）小于右侧的数字（除数）**，且两者均为正数，那么结果始终等于 ** 左侧的数字本身 **。

选项分析：

A. **ValueError：**错误。取模运算对整数是完全合法的。

B. **0.33：**错误。这是除法的结果，不是余数。

C. **2：正确**。2 除以 6 商 0 余 2。

结论： `2 % 6` 的结果是 **2**。故正确答案是 **C**。

36. What is the output of the following code? **B**

```
x = 6
y = 2
print(x ** y)
```

A. 66　B. 36

详细解答

这道题考察的是 Python 中的 **幂运算符 (Exponentiation Operator)**，即 `**`。

- **运算符含义**：在 Python 中，`**` 用于计算底数的指数次幂。表达式 `x ** y` 等价于数学中的 x^y。
- **变量赋值**：
  - `x = 6`（底数）
  - `y = 2`（指数）

计算过程：

(a) 将变量代入表达式：`6 ** 2`。

(b) 计算 6 的平方：$6 \times 6 = 36$。

选项分析：

A. **66**：错误。这可能是误将两个数字连起来写，或者是对运算符理解不正确。

B. **36**：**正确**。这是 6 的 2 次方的计算结果。

知识对比：

- `x * y` 的结果是 12（乘法）。
- `x ** y` 的结果是 36（幂运算）。
- 在某些其他语言（如 C++ 或 Java）中，幂运算通常使用 `pow(x, y)` 函数，而在 Python 中使用 `**` 更加简洁直观。

结论：该代码输出底数的平方值。故正确答案是 **B**。

37. What is the output of the following code? **B**

```
x = 6
y = 2
print(x // y)
```

A. 0　B. 3

详细解答

这道题考察的是 Python 中的 **整除运算符 (Floor Division)**，符号为 `//`。

- **运算符定义：**`//` 执行除法运算，但会向下取整到最接近的整数（即“地板除”），返回结果的整数部分。
- **与真除法的区别：**
  - `/` (单斜杠)：返回浮点数结果（如 `6 / 2` 得到 `3.0`）。
  - `//` (双斜杠)：返回整数结果（如 `6 // 2` 得到 `3`）。

逻辑推导：

(a) 变量代入：执行 `6 // 2`。

(b) 数学计算：$6 \div 2 = 3$。

(c) 类型处理：由于使用的是整除运算符 `//`，结果不带小数位，直接输出 **3**。

选项分析：

A. **0**：错误。这是在除数远大于被除数（且不进行浮点运算）时可能出现的逻辑错误。

B. **3**：**正确**。这是整数除法的精确结果。

结论： `x // y` 在此例中输出整数 3。故正确答案是 **B**。

进阶提醒：如果运算中包含负数，结果会向负无穷方向取整。例如 `-7 // 2` 的结果是 `-4` 而不是 `-3`。

38. What is the output of the following Python code? **C**

```
x = 10
y = 50
if x ** 2 > 100 and y < 100:
    print(x, y)
```

A. 100 500　B. 10 50　C. None

详细解答

这道题考察的是 **条件判断语句 (if statement)** 中的 **布尔逻辑运算符 (and)** 以及 **算术优先级**。

- **逻辑规则：**对于 `A and B` 表达式，只有当条件 A 和条件 B 同时为 `True` 时，整个条件才成立。
- **短路逻辑：**如果第一个条件 A 为 `False`，Python 甚至不会去检查条件 B，直接判定整体为 `False`。

条件拆解分析：已知变量：`x = 10`, `y = 50`。

(a) **检查第一个条件：**`x ** 2 > 100`

- 计算 $10^2 = 100$。
- 判断 $100 > 100$。
- 结果为 **`False`**（因为 100 等于 100，并不大于 100）。

(b) **整体判定：**

- 既然第一个条件已经失败 (`False`)，无论 `y < 100` 是否成立，整个 `if` 条件都判定为 **`False`**。

执行结果：

- 由于 `if` 条件不成立，内部的 `print(x, y)` 语句 ** 永远不会被执行 **。
- 控制台不会输出任何内容。

结论：该代码运行后没有输出。在选择题语境下，表示“无输出”或“没有执行”的选项通常标记为 **None**。故正确答案是 **C**。

39. What is the value of the variable `var` after the for loop completes its execution? **A**

```
var = 10
for i in range(10):
    for j in range(2, 10, 1):
        if var % 2 == 0:
            continue
```

```
            var += 1
    var += 1

print(var)
```

A. 20　B. 21　C. 10　D. 30

详细解答

这道题考察的是 **循环结构**、**continue 关键字**以及 **代码缩进（不可达代码）**的综合理解。

- **continue 的作用**：一旦执行到 continue，程序会立即跳过当前循环体中剩余的所有语句，直接进入下一次循环迭代。
- **不可达代码 (Unreachable Code)**：在 continue 之后的同一缩进层级的代码永远不会被执行。

逻辑分析：

(a) **初始化**：var = 10。

(b) **外层循环**：运行 10 次（i 从 0 到 9）。

(c) **内层循环**：运行 8 次（j 从 2 到 9）。

(d) **关键条件判断**：

- 初始时 var = 10。在内层循环中，var % 2 == 0 成立。
- 执行 continue。
- **注意**：原本打算执行的 var += 1（在 continue 下方）被永远跳过了。
- 因此，内层循环跑了 8 次，但对 var 没有任何改变。

(e) **外层步进**：内层循环结束后，执行外层的 var += 1。

计算过程：

- 外层循环第 1 次：var 变为 11。
- 外层循环第 2 次：内层循环中 var % 2 == 0 为假（因为 11 是奇数），所以不会执行 continue。

- **陷阱：**虽然不执行 `continue`，但由于代码逻辑，`var += 1` 依然在外层循环中执行。
- **事实：**无论内层循环发生什么，`var` 每一轮外层循环只会在末尾雷打不动地增加 1。

由于外层循环共执行 **10 次**，`var` 从 10 开始，每次加 1：

$$10 + (1 \times 10) = 20$$

结论：变量 `var` 的最终值是 **20**。故正确答案是 **A**。

40. What is the value of `x`? **D**

```
x = 0
while (x < 100):
    x += 2

print(x)
```

A. 101　B. 99　C. None of the above, this is an infinite loop　D. 100

详细解答

这道题考察的是 **`while` 循环**的执行逻辑以及 **循环终止条件**的判定。

- **循环逻辑：**`while` 循环会在条件（`x < 100`）为真时持续执行其内部的代码块。
- **步进方式：**每次循环 `x` 都会增加 2（即 `x = x + 2`）。

关键执行步骤分析：

(a) **初始状态：**`x = 0`。

(b) **循环过程中：**`x` 会经历 `0, 2, 4, ..., 96, 98`。

(c) **临界点判断：**

- 当 `x = 98` 时，条件 `98 < 100` 为 **真**，进入循环。
- 执行 `x += 2`，此时 `x` 变为 **100**。

(d) **终止判定：**

- 回到循环顶部，判断条件 `100 < 100`。
- 结果为 **假 (False)**，循环立即停止。

选项分析：

A. **101**：错误。步进是 2，且 101 不满足停止逻辑。

B. **99**：错误。步进是 2，`x` 始终为偶数。

C. **无尽循环**：错误。由于 `x` 在不断增加，最终一定会达到并超过 100。

D. **100**：**正确**。它是第一个使 `x < 100` 变为假的偶数。

结论：当 `x` 增加到 100 时，条件不再满足，循环退出并打印当前值。故正确答案是 **D**。

41. What is the output of the following if statement?

```
a, b = 12, 5
if a + b:
    print('True')
else:
    print('False')
```

A. False　B. True

详细解答

这道题考察的是 Python 中 ** 隐式布尔转换 (Implicit Boolean Conversion)** 的原则。

- **算术运算：**首先计算 `a + b` 的值。$12 + 5 = 17$。
- **布尔评估：**在 `if` 语句中，Python 会自动调用 `bool()` 函数来评估表达式的真假。

逻辑核心：在 Python 中，对于数值类型：

- **0** (整数零)、**0.0** (浮点数零) 被视为 **`False`**。
- **任何非零数值**（无论是正数还是负数）都被视为 **`True`**。

执行流程分析：

(a) 变量赋值：`a = 12, b = 5`。

(b) 计算 `a + b` 得到 `17`。

(c) 执行 `if 17:`。

(d) 由于 `17` 是一个非零整数，条件评估为 `True`。

(e) 程序执行 `if` 分支下的代码，打印字符串 `'True'`。

结论：该代码会进入 `if` 分支。故正确答案是 **B**。

42. Given the nested if-else below, what will be the value of `x` when the code executed successfully? **D**

```
x = 0
a = 5
b = 5
if a > 0:
    if b < 0:
        x = x + 5
    elif a > 5:
        x = x + 4
    else:
        x = x + 3
else:
    x = x + 2
print(x)
```

A. 0　B. 4　C. 2　D. 3

详细解答

这道题考察的是 **嵌套条件语句 (Nested if-else)** 的逻辑流向。解决这类问题的关键是看清每一层缩进对应的判断条件。

- **初始化：**`x = 0, a = 5, b = 5`。

逻辑步进分析：

(a) **第一层判断：** `if a > 0:`

- 因为 `5 > 0` 为 **True**，程序进入外层 `if` 块。
- 此时，最底部的 `else: x = x + 2` 将会被跳过，永远不会执行。

(b) **第二层判断（嵌套内层）：**

- **首先检查** `if b < 0:`
  - 因为 `5 < 0` 为 **False**，跳过此分支。
- **接着检查** `elif a > 5:`
  - 因为 `5 > 5` 为 **False**，跳过此分支。
- **进入最后的** `else:` **分支：**
  - 前两个内层条件均不满足，执行 `x = x + 3`。

最终计算：

- `x` 原本是 `0`。
- 执行 `x = 0 + 3`。
- 最终打印结果为 **3**。

结论：程序精确命中了内层结构的 `else` 分支。故正确答案是 **D**。

43. What is the output of the following loop? **B**

```
for l in 'Jhon':
    if l == 'o':
        pass
    print(l, end=", ")
```

A. J, h, n, B. J, h, o, n,

详细解答

这道题考察的是 Python 中的 **pass** 语句及其与 `continue` 的区别。

- **pass 语句的本质：** pass 是一个 **空操作 (Null Operation)**。它被用作占位符，当语法上需要一个语句但程序不需要执行任何操作时使用。

- **对执行流的影响：**执行 pass 之后，程序会继续按照正常的顺序执行后续的代码，** 不会 ** 跳过当前的循环迭代，也 ** 不会 ** 终止循环。

逻辑步进分析：

(a) **迭代 1：** `l = 'J'`。条件 `'J' == 'o'` 为假。打印 `J, `。

(b) **迭代 2：** `l = 'h'`。条件 `'h' == 'o'` 为假。打印 `h, `。

(c) **迭代 3：** `l = 'o'`。条件 `'o' == 'o'` 为真。

- 执行 pass（什么都不做）。
- 程序继续向下执行 `print(l, end=", ")`。
- 打印 `o, `。

(d) **迭代 4：** `l = 'n'`。条件 `'n' == 'o'` 为假。打印 `n, `。

关键对比：

- 如果使用 **continue：**遇到 `'o'` 时会跳过打印，输出将是 `J, h, n, `（即选项 A）。
- 因为使用的是 **pass：**代码逻辑不受任何影响，所有的字符都会被原样打印。

结论： pass 不会阻止后续代码的执行。故正确答案是 **B**。

44. What is the output of the following list assignment? **B**

```
aList = [4, 8, 12, 16]
aList[1:4] = [20, 24, 28]
print(aList)
```

A. [4, 20, 24, 28, 8, 12, 16]

B. [4, 20, 24, 28]

详细解答

这道题考察的是 Python 列表的 **切片赋值 (Slice Assignment)** 机制。

- **切片范围：** `aList[1:4]` 表示选取索引从 1 开始到 4 结束（但不包括 4）的元素。
  - 索引 1 是 8
  - 索引 2 是 12
  - 索引 3 是 16
  - 因此，被选中的部分是 `[8, 12, 16]`。
- **替换逻辑：** 切片赋值会移除选中的部分，并用赋值号右侧的可迭代对象（本题中为 `[20, 24, 28]`）替换该位置。

执行过程分析：

(a) 初始列表：`[4, 8, 12, 16]`

(b) 确定替换区间：从索引 1 到末尾的部分 `[8, 12, 16]` 被整体“挖掉”。

(c) 插入新内容：在被挖掉的位置填入新列表 `[20, 24, 28]`。

(d) 最终组合：首位的 4 保持不变，后面紧跟新插入的三个元素。

选项分析：

A. **[4, 20, 24, 28, 8, 12, 16]**：错误。这是在索引 1 位置执行了插入操作（如 `insert`），而不是切片替换。

B. **[4, 20, 24, 28]**：**正确**。旧元素被新元素完全覆盖。

知识拓展： 如果右侧列表的长度与切片长度不等，列表的总长度会发生变化。例如，如果执行 `aList[1:4] = [100]`，结果将变为 `[4, 100]`。

结论： 切片赋值是“原地替换”行为。故正确答案是 **B**。

45. What is the output of the following list comprehension? **A**

```
resList = [x+y for x in ['Hello ', 'Good '] for y in ['Dear',
    'Bye']]
print(resList)
```

A. [‘Hello Dear’, ‘Hello Bye’, ‘Good Dear’, ’Good Bye’]
B. [‘Hello Dear’, ‘Good Dear’, ‘Hello Bye’, ’Good Bye’]

详细解答

这道题考察的是 **嵌套列表推导式 (Nested List Comprehension)** 的执行顺序。

- **核心规则：**在列表推导式中，多个 `for` 子句的执行顺序是从左到右的，这与嵌套循环的逻辑完全一致。
- **等价逻辑：**代码可以转换为以下传统的嵌套循环形式：

```
resList = []
for x in ['Hello ', 'Good ']:       # 外层循环
    for y in ['Dear', 'Bye']:       # 内层循环
        resList.append(x + y)
```

执行过程分析：

(a) **固定外层第一个值：** `x = 'Hello '`

- 遍历内层第一个值：`y = 'Dear'` → `'Hello Dear'`
- 遍历内层第二个值：`y = 'Bye'` → `'Hello Bye'`

(b) **固定外层第二个值：** `x = 'Good '`

- 遍历内层第一个值：`y = 'Dear'` → `'Good Dear'`
- 遍历内层第二个值：`y = 'Bye'` → `'Good Bye'`

选项分析：

A. **正确排列：**遵循了“外层不动，内层走完”的原则。先处理完所有 `Hello` 的组合，再处理 `Good` 的组合。

B. **错误排列：**这是交叉排列，不符合 Python 从左至右解析 `for` 语句的顺序。

结论：列表推导式的顺序决定了结果的排列。故正确答案是 **A**。

46. What is the output of the following list operation? **A**

```
sampleList = [10, 20, 30, 40, 50]
print(sampleList[-2])
```

A. 40　B. 20　C. IndexError: list index out of range

详细解答

这道题考察的是 Python 列表中的 **负数索引 (Negative Indexing)** 机制。

- **正数索引**：从左向右计数，起始索引为 0。
- **负数索引**：从右向左计数，起始索引为 -1（即列表的最后一个元素）。

逻辑拆解分析：对于列表 [10, 20, 30, 40, 50]：

- 索引 -1 对应最后一个元素：50。
- 索引 -2 对应倒数第二个元素：40。
- 索引 -3 对应倒数第三个元素：30，依此类推。

选项分析：

A. **40**：**正确**。这是倒数第二个位置的元素。

B. **20**：错误。这是正数索引 1 对应的元素。

C. **IndexError**：错误。只要索引的绝对值不超过列表长度，负数索引就是合法的。

结论： sampleList[-2] 访问的是倒数第二个元素。故正确答案是 **A**。

47. What is the output of the following list operation? **A**

```
sampleList = [10, 20, 30, 40, 50]
print(sampleList[-4:-1])
```

A. [20, 30, 40]　B. [40, 30, 20]　C. IndexError: list index out of range

详细解答

这道题结合了 **负数索引 (Negative Indexing)** 和 **切片 (Slicing)** 两个核心知识点。

- **切片语法**：`list[start:stop:step]`。
  - `start`：切片开始的位置（包含）。
  - `stop`：切片结束的位置（不包含）。
  - `step`：默认为 1，表示从左向右提取。

逻辑拆解分析：对于列表 `[10, 20, 30, 40, 50]`，我们先标定负数索引的位置：

(a) **起始点 (`start = -4`)**：对应元素 **20**。

(b) **终止点 (`stop = -1`)**：对应元素 **50**。注意，切片是“左闭右开”的，所以不包含 `-1` 位置的 `50`。

(c) **步长 (`step`)**：由于没有指定，默认为 `1`，即向右移动。

从 `-4` 位置开始，向右提取到 `-1` 之前，结果为：`[20, 30, 40]`。

选项分析：

A. **[20, 30, 40]**：**正确**。符合从索引 `-4` 到 `-1` 的提取逻辑。

B. **[40, 30, 20]**：错误。这通常出现在步长为 `-1`（逆序）时，但本题步长为正。

C. **IndexError**：错误。切片操作在 Python 中非常鲁棒，即便索引越界通常也只返回空列表，而不会抛出错误。

结论：该切片选取了列表的第 2、3、4 个元素。故正确答案是 **A**。

48. What is the output of the following code? **A**

```
sampleList = [10, 20, 30, 40]
del sampleList[0:6]
print(sampleList)
```

A. [ ]　B. list index out of range　C. [10, 20]

详细解答

这道题考察的是 Python 中 `del` 关键字配合 **切片 (Slicing)** 时的行为，以及 Python 对 **切片索引越界**的处理机制。

- **`del` 的作用：**用于删除对象。当作用于列表切片时，它会从原列表中移除该切片覆盖的所有元素。
- **切片的鲁棒性：**在 Python 中，切片索引（如 `0:6`）即使超出了列表的实际长度（本例长度为 4），程序 ** 不会报错 **，而是会自动处理为“到列表末尾为止”。

执行过程分析：

(a) **初始列表：**`sampleList = [10, 20, 30, 40]`，索引范围为 `0` 到 `3`。

(b) **切片解析：**`sampleList[0:6]` 尝试获取索引从 `0` 到 `5` 的元素。

(c) **边界处理：**由于列表在索引 `3` 处就结束了，Python 会将切片实际识别为 `[10, 20, 30, 40]`（即整个列表）。

(d) **执行删除：**`del` 指令将这部分内容全部移除。

选项分析：

A. `[ ]`：**正确**。整个列表的元素都在切片范围内，被全部清空，留下一个空列表。

B. **list index out of range：**错误。这是普通索引（如 `sampleList[6]`）会抛出的错误，但切片操作永远不会因为越界而报错。

C. **[10, 20]**：错误。这没有任何逻辑依据，除非切片范围是 `2:4`。

结论： `del` 配合越界切片会清空所有匹配到的元素。故正确答案是 **A**。

49. What is the output of the following list function? **B**

```
sampleList = [10, 20, 30, 40, 50]
sampleList.append(60)
print(sampleList)
```

A. [10, 20, 30, 40, 50]　B. [10, 20, 30, 40, 50, 60]

详细解答

这道题考察的是 Python 列表（list）最常用的内置方法之一：**append()**。

- **append() 的功能**：将一个对象（作为单个元素）添加到列表的 **末尾 (End)**。
- **原地修改 (In-place)**：append() 方法会直接修改原始列表，而不返回新列表。

执行过程分析：

(a) **初始状态**：sampleList 包含 5 个元素 [10, 20, 30, 40, 50]。

(b) **执行操作**：调用 sampleList.append(60)。

(c) **结果**：数字 60 被放置在索引 5 的位置（紧跟在 50 之后）。

选项分析：

A. **[10, 20, 30, 40, 50]**：错误。这表示列表没有发生变化，忽略了 append 的作用。

B. **[10, 20, 30, 40, 50, 60]**：**正确**。列表长度从 5 增加到了 6，且新元素位于末尾。

知识拓展：如果要一次性添加多个元素（合并列表），应使用 extend() 方法。如果此时执行的是 sampleList.append([70, 80])，结果会变成 [..., 60, [70, 80]]，即将整个子列表作为一个元素添加。

结论： append() 始终在末尾追加元素。故正确答案是 **B**。

50. What is the output of the following list operation? **D**

```
aList = [10, 20, 30, 40, 50, 60, 70, 80]
print(aList[2:5])
```

A. [20, 30, 40, 50]　B. [10, 20, 30, 40]　C. [30, 40, 50, 60, 70, 80]　D. [30, 40, 50]

详细解答

这道题考察的是 Python 列表的 ** 标准切片 (Standard Slicing)** 操作。

- **切片语法：** `aList[start:stop]`。
- **包含规则：**
  - `start` (索引 2)：** 包含 ** 起始位置的元素。
  - `stop` (索引 5)：** 不包含 ** 结束位置的元素（即取到 `stop-1` 为止）。

逻辑拆解分析：对于列表 `[10, 20, 30, 40, 50, 60, 70, 80]`：

(a) **定位索引 0：** `10`

(b) **定位索引 1：** `20`

(c) **起始位置 (索引 2)：**元素是 **30**（从这里开始提取）。

(d) **中间元素 (索引 3)：**元素是 **40**。

(e) **中间元素 (索引 4)：**元素是 **50**。

(f) **终止位置 (索引 5)：**元素是 `60`（** 不包含 ** 此元素，在此处停止）。

提取出的元素序列为：`[30, 40, 50]`。

选项分析：

A. **[20, 30, 40, 50]：**错误。这是从索引 1 开始的结果。

B. **[10, 20, 30, 40]：**错误。这是从索引 0 开始且长度为 4 的结果。

C. **[30, ... , 80]：**错误。这是忽略了 `stop` 参数的结果。

D. **[30, 40, 50]：正确**。严格符合索引 2 到 4 的范围。

结论：切片遵循“左闭右开”原则。故正确答案是 **D**。

51. What is the output of the following list operation? **B**

```
aList = [10, 20, 30, 40, 50, 60, 70, 80]
print(aList[:4])
```

A. [20, 30, 40, 50]
B. [10, 20, 30, 40]
C. [30, 40, 50, 60, 70, 80]
D. [30, 40, 50]
E。[40, 50, 60, 70, 80]

详细解答

这道题考察的是 Python 列表切片中 ** 省略起始索引 (Omitted Start Index)** 的默认行为。

- **切片语法：**`aList[start:stop]`。
- **默认值规则：**
  - 如果 `start` 被省略（如 `[:4]`），Python 默认从索引 **0**（即列表的开头）开始提取。
  - `stop` (索引 4)：依然遵循“左闭右开”原则，即 ** 不包含 ** 索引为 4 的元素。

逻辑拆解分析：对于列表 `[10, 20, 30, 40, 50, 60, 70, 80]`：

(a) **确定起始点：**由于冒号左侧为空，从索引 **0** 开始 → 元素为 **10**。

(b) **确定终止点：**停止于索引 **4**（对应元素 50），但不包含它。

(c) **提取区间：**提取索引 0, 1, 2, 3 对应的元素。

提取出的元素序列为：`[10, 20, 30, 40]`。

选项分析：

A. **[20, 30, 40, 50]：**错误。这是索引从 1 到 5 的结果。

B. **[10, 20, 30, 40]：正确**。从头开始取 4 个元素。

C. **[30, 40, 50, 60, 70, 80]：**错误。这是省略结束索引且从索引 2 开始的结果。

D. **[30, 40, 50]：**错误。这是上道题（2:5）的结果。

E. **[40, 50, 60, 70, 80]：**错误。这是从索引 3 开始到结束的结果。

结论： `[:n]` 的快捷含义是“获取列表的前 n 个元素”。故正确答案是 **B**。

52. What is the output of the following list operation? **D**

```
aList = [10, 20, 30, 40, 50, 60, 70, 80]
print(aList[3:])
```

A. [20, 30, 40, 50]
B. [30, 40, 50, 60, 70, 80]
C. [10, 20, 30, 40]
D. [40, 50, 60, 70, 80]

详细解答

这道题考察的是 Python 列表切片中 ** 省略结束索引 (Omitted Stop Index)** 的默认行为。

- **切片语法：**`aList[start:stop]`。
- **默认值规则：**
  - **起始位置 (start)：**本题为 3，即从索引为 3 的元素开始（包含该元素）。
  - **结束位置 (stop)：**如果冒号右侧被省略（如 `[3:]`），Python 默认提取到列表的 **最后一个元素（包含末尾）**。

逻辑拆解分析：对于列表 `[10, 20, 30, 40, 50, 60, 70, 80]`：

(a) **定位索引 0, 1, 2：**分别是 `10`，`20`，`30`。

(b) **确定起始点 (索引 3)：**对应的元素是 **40**。

(c) **确定范围：**由于没有指定终止索引，切片会从 `40` 开始，一直抓取到列表末尾的 `80`。

提取出的元素序列为：`[40, 50, 60, 70, 80]`。

选项分析：

A. **[20, 30, 40, 50]：**错误。这是 `aList[1:5]` 的结果。

B. **[30, 40, 50, 60, 70, 80]:**错误。这是从索引 2 开始的结果 (`aList[2:]`)。

C. **[10, 20, 30, 40]：**错误。这是获取前 4 个元素的结果 (`aList[:4]`)。

D. **[40, 50, 60, 70, 80]：正确。**从索引 3 开始取到末尾。

结论： `aList[n:]` 的含义是“获取从索引 n 到最后的所有元素”。故正确答案是 **D**。

53. What is the output of the following code? **B**

```
list1 = ['xyz', 'zara', 'Beijing-Dublin']
print(max(list1))
```

A. Beijing-Dublin　B. zara　C. xyz

详细解答

这道题考察的是 Python 内置函数 **`max()`** 作用于字符串列表时的比较逻辑。

- **比较原则：** 当 `max()` 用于字符串时，它不是比较字符串的长度，而是按照 **字典序 (Lexicographical Order)** 进行比较。
- **底层机制：** Python 会逐个字符地比较字符串的 **Unicode 编码值**（即 `ord()` 函数返回的数值）。

逻辑拆解分析： 我们来比较列表中三个字符串的首字母：

(a) `'xyz'` 的首字母是 **'x'** (Unicode: 120)

(b) `'zara'` 的首字母是 **'z'** (Unicode: 122)

(c) `'Beijing-Dublin'` 的首字母是 **'B'** (Unicode: 66)

判定过程：

- 首先，大写字母的编码值小于小写字母，所以 `'Beijing-Dublin'` 排在最前面（最小）。
- 接着比较 `'x'` 和 `'z'`。在字母表中，`'z'` 出现在 `'x'` 之后，因此其编码值更大。
- 既然 `'z'` 是最大的首字母，`'zara'` 就是这组字符串中字典序最大的元素。

选项分析：

A. **Beijing-Dublin：** 错误。它虽然包含最多字符（长度最长），但首字母'B' 的 Unicode 值最小。

B. **zara：正确**。首字母'z' 的字典序最大。

C. **xyz**：错误。'x' 小于'z'。

结论：字符串比较看重的是字符编码顺序而非长度。故正确答案是 **B**。

54. What is the output of the following? **A**

```
l = ["a"] * 10
print(len(l))
```

A. 10　B. 0　C. Syntax Error

详细解答

这道题考察的是 Python 列表的 **乘法运算符 (*)**，即 **重复 (Repetition)** 操作。

- **列表重复**：当列表与整数 n 相乘时，Python 会创建一个新列表，其中包含原列表元素的 n 次重复。
- **`len()` 函数**：用于返回对象中元素的个数。

逻辑拆解分析：

(a) **初始化**：原列表为 `["a"]`，长度为 1。

(b) **执行乘法**：`["a"] * 10`。

- 结果为 `['a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a', 'a']`。

(c) **计算长度**：`len(l)` 统计该列表中的元素总数，结果自然为 10。

选项分析：

A. **10**：**正确**。原列表的一个元素被重复了 10 次。

B. **0**：错误。列表不为空。

C. **Syntax Error**：错误。列表乘法是 Python 合法且常用的语法。

注意陷阱：如果是 `l = "a" * 10`（字符串乘以 10），其长度同样是 10。但如果原列表包含多个元素，例如 `[1, 2] * 3`，结果会是 `[1, 2, 1, 2, 1, 2]`，长度则为 6。

结论：乘法操作扩展了列表的规模。故正确答案是 **A**。

55. Select all the correct options to join two lists in Python: **A, C**

```
listOne = ['a', 'b', 'c', 'd']
listTwo = ['e', 'f', 'g']
```

A. `newList = listOne + listTwo`

B. `newList = listOne * listTwo`

C. `newList = listOne.extend(listTwo)`

D. `newList.extend(listOne, listTwo)`

详细解答

这道题考察的是 Python 中合并两个列表的不同方法以及它们的返回值特性。

- **运算符重载 (+)：**用于合并两个列表并返回一个包含两者元素的新列表。
- **`extend()` 方法：**用于将一个列表中的所有元素添加到另一个列表的末尾。这是一个 **原地 (In-place)** 修改操作。

选项逐一分析：

A. **`newList = listOne + listTwo`:**

- **正确**。这是合并列表最直观的方法。它不修改原有的 `listOne`，而是产生一个新的对象 `newList`。

B. **`newList = listOne * listTwo`:**

- **错误**。列表不支持“乘法”运算（除非第二个操作数是整数，表示重复）。两个列表相乘会抛出 `TypeError`。

C. **`listOne.extend(listTwo)`:**

- **技术性正确 (但需注意赋值陷阱)**。题干问的是“join two lists”的方法。使用 `listOne.extend(listTwo)` 确实成功地将两个列表合并到了 `listOne` 中。
- **注意：**代码中写作 `newList = listOne.extend(listTwo)` 是一个常见错误，因为 `extend()` 返回 `None`。所以虽然“合并”成功了，但 `newList` 的值会是 `None`。

D. **`newList.extend(listOne, listTwo)`:**

- **错误**。`extend()` 方法仅接收一个参数（一个可迭代对象）。同时，如果 `newList` 尚未定义，会抛出 `NameError`。

结论：选项 A 是创建新列表的标准做法，选项 C 是原地扩展的标准做法。故正确答案是 **A, C**。

56. In Python, list is mutable: **B**

A. False B. True

详细解答

这道题考察的是 Python 数据结构中最核心的概念之一：**可变性 (Mutability)**。

- **可变对象 (Mutable)：**在创建后，其内部状态或内容可以被修改，而无需创建新对象。
- **不可变对象 (Immutable)：**一旦创建，其内容就不能更改。如果尝试修改，Python 会创建一个新的对象。

知识点拆解：

(a) **列表的特性：**列表（`list`）是典型的 **可变对象**。你可以执行以下操作而保持对象的 ID（内存地址）不变：

- 修改单个元素：`L[0] = 99`
- 添加元素：`L.append(4)`
- 删除元素：`del L[1]`

(b) **对比不可变对象：**

- **数字** (int, float), **字符串** (str), **元组** (tuple) 都是不可变的。

结论：这是一个基础概念题。列表（List）、字典（Dict）和集合（Set）是 Python 的三大主要可变类型。

57. What is the output of the following list function? **B**

```
sampleList = [10, 20, 30, 40, 50]
sampleList.pop()
print(sampleList)
```

A. [20, 30, 40, 50]　B. [10, 20, 30, 40]　C. [10, 20, 30, 40]

详细解答

这道题考察的是 Python 列表（list）的内置方法 `pop()` 的默认行为。

- **`pop()` 的定义：**用于移除列表中的一个元素，并返回该元素的值。
- **默认索引：**如果调用时没有传递参数（即空括号 `()`），它默认移除并返回列表中的 **最后一个元素**（索引为 `-1`）。

执行过程分析：

(a) **初始列表：**`sampleList = [10, 20, 30, 40, 50]`。

(b) **执行 `pop()`：**

- Python 定位到列表末尾的元素 `50`。
- 将其从内存中从列表中移除。

(c) **打印结果：**列表现在只剩下前四个元素 `[10, 20, 30, 40]`。

知识拓展：

- 如果你想移除特定的元素，可以传递索引，例如 `sampleList.pop(0)` 会移除并返回 `10`。
- **对比 `remove()`：**`remove(x)` 是根据 **值**来删除（删除第一个匹配到的值），而 `pop(i)` 是根据 **索引**来删除。

结论：不带参数的 `pop()` 相当于 `pop(-1)`。故正确答案是 **B**（注意选项 C 与 B 内容一致，通常选择第一个匹配的正确项）。

58. What is the output of the following list function? **B**

```
sampleList = [10, 20, 30, 40, 50]
sampleList.pop(2)
print(sampleList)
```

A. [10, 20, 30, 40, 50]　B. [10, 20, 40, 50]　C. [10, 20, 30, 40]　D. [10, 20, 30, 50]

详细解答

这道题考察的是 `pop()` 方法在指定 **索引 (Index)** 时的行为。

- **`pop(index)`：** 移除并返回列表中指定索引位置的元素。
- **索引规则：** Python 的索引从 0 开始计数。

执行过程分析： 对于列表 `[10, 20, 30, 40, 50]`：

(a) **定位索引：**

- 索引 0: 10
- 索引 1: 20
- **索引 2: 30** ← 这是被选中的目标。

(b) **执行删除：** 调用 `sampleList.pop(2)` 会将元素 30 从列表中永久移除。

(c) **后续移动：** 原本在索引 3 和 4 的元素（40 和 50）会向左移动，填补空位。

选项分析：

A. **[10, 20, 30, 40, 50]：** 错误。这是没有执行任何删除操作的原列表。

B. **[10, 20, 40, 50]：正确。** 元素 30（索引 2）已被移除。

C. **[10, 20, 30, 40]：** 错误。这是执行了 `pop()`（默认末尾）的结果。

D. **[10, 20, 30, 50]：** 错误。这是移除了索引 3（元素 40）的结果。

结论： `pop(2)` 指向的是列表中的第三个元素。故正确答案是 **B**。

59. What is the output of the following code? **B**

```
aList = ["Beijing-Dublin", [4, 8, 12, 16]]
print(aList[0][1])
```

A. B

B. e

C. i

D. j

详细解答

这道题考察的是 Python 中的 **多维索引 (Multi-dimensional Indexing)**，即如何访问嵌套在列表中的序列元素。

- **第一层索引 aList[0]：**获取列表中的第一个元素。在本例中，第一个元素是字符串 "Beijing-Dublin"。
- **第二层索引 [1]：**在第一层获取到的字符串基础上，获取该字符串中索引为 1 的字符。

逻辑拆解分析：

(a) **定位 aList[0]：**

- aList 是 ["Beijing-Dublin", [4, 8, 12, 16]]。
- 索引 0 指向字符串 "Beijing-Dublin"。

(b) **定位字符串内部索引：**对于字符串 "Beijing-Dublin"，其索引分布如下：

- 索引 0 : B
- **索引 1 : e** ← 目标字符
- 索引 2 : i
- 索引 3 : j

选项分析：

A. **B：**这是 aList[0][0]。

B. **e：正确**。它是字符串中第二个字符。

C. **i：**这是 aList[0][2]。

D. **j**：这是 `aList[0][3]`。

知识拓展：如果我们要访问嵌套列表里的数字 `8`，代码应该是 `aList[1][1]`。

结论：连续索引是按照从外向里的顺序解析的。故正确答案是 **B**。

60. What is the output of the following code? **C**

```
aList = ["Beijing-Dublin", [4, 8, 12, 16]]
print(aList[1][3])
```

A. 8

B. 12

C. 16

D. 4

详细解答

这道题考察的是 Python 中对 **嵌套列表 (Nested List)** 的访问能力，即通过多级索引获取内部序列的值。

- **第一层索引 `aList[1]`**：获取外层列表中的第二个元素。在本例中，索引 `0` 是字符串，索引 `1` 是子列表 `[4, 8, 12, 16]`。
- **第二层索引 `[3]`**：在获取到的子列表内部，提取索引为 `3` 的元素。

逻辑拆解分析：

(a) **定位外层索引：**

- `aList[0]` → `"Beijing-Dublin"`
- `aList[1]` → `[4, 8, 12, 16]`

(b) **定位子列表内部索引：**针对子列表 `[4, 8, 12, 16]`，其索引分布如下：

- 索引 `0` : `4`
- 索引 `1` : `8`
- 索引 `2` : `12`
- **索引 `3` : `16`** ← 目标数值

选项分析：

A. **8**：这是 `aList[1][1]`。

B. **12**：这是 `aList[1][2]`。

C. **16**：**正确**。它是子列表中索引为 3 的元素。

D. **4**：这是 `aList[1][0]`。

结论：通过 `aList[1][3]`，我们成功进入了嵌套列表并取到了最后一个整数。故正确答案是 **C**。

61. What is the output of the following: **C**

```
aList = [5, 10, 15, 25]
print(aList[::-2])
```

A. `[15, 10, 5]`

B. `[10, 5]`

C. `[25, 10]`

详细解答

这道题考察的是 Python 列表的 **切片 (Slicing)** 操作，特别是带有 **负数步长 (Negative Step)** 的用法。

- **语法结构 `list[start:stop:step]`**：在本例中，`start` 和 `stop` 都被省略了。
- **步长 `-2`**：步长为负数时，切片会从右向左（反向）进行提取。

逻辑拆解分析：

(a) **确定起始点**：当步长为负数且未指定 `start` 时，默认从列表的 **最后一个元素**开始。

- 当前列表末尾元素为 25 (索引 3)。

(b) **执行跳步逻辑：**

- **第 1 个提取值**：索引 3 → 25
- **第 2 个提取值**：跳过一个元素（索引 2），移动到索引 `3 - 2 = 1` → 10

- **终止条件：**继续移动会超出列表起始边界，提取停止。

选项分析：

A. **[15, 10, 5]：**这是 `aList[2::-1]` 的结果。

B. **[10, 5]：**这是 `aList[1::-1]` 的结果。

C. **[25, 10]：正确。**从末尾 25 开始反向每隔一个元素取值。

结论：切片 `[::-2]` 的效果是“反向取值，步长为 2”。结果构成的子列表为 `[25, 10]`。故正确答案是 **C**。

62. What is the output of the following: **A**

```
aList = [1, 2, 3, 4, 5, 6, 7]
pow2 = [2 * x for x in aList]
print(pow2)
```

A. `[2, 4, 6, 8, 10, 12, 14]`

B. `[2, 4, 8, 16, 32, 64, 128]`

详细解答

这道题考察的是 Python 中的 **列表推导式 (List Comprehension)**。这是一种简洁的创建列表的方法，其基本语法为 `[expression for item in iterable]`。

- **遍历对象 (`in aList`)：**程序会依次取出 `aList` 中的每一个元素。
- **操作逻辑 (`2 * x`)：**对于每一个取出的元素 `x`，都会执行乘以 2 的操作。
- **结果生成：**计算后的新值会被存入一个新的列表 `pow2` 中。

逻辑拆解分析：

(a) **循环过程：**

- 当 `x = 1` 时，`2 * 1 = 2`
- 当 `x = 2` 时，`2 * 2 = 4`

- 当 `x = 3` 时，`2 * 3 = 6`
- ……以此类推，直到最后一个元素 7。
- 当 `x = 7` 时，`2 * 7 = 14`

(b) **组合结果**：最终生成的列表为 `[2, 4, 6, 8, 10, 12, 14]`。

选项分析：

A. **[2, 4, 6, 8, 10, 12, 14]**：**正确**。每个元素都成功乘以了 2。

B. **[2, 4, 8, 16, 32, 64, 128]**：这是计算 2^x（即 `2 ** x`）的结果。虽然变量名为 `pow2`（通常暗示幂运算），但代码中实际使用的是乘法运算符 `*`。

结论：题目中的代码执行的是“数值翻倍”而非“求 2 的幂”。在编程考试中，要注意区分运算符 `*` (乘法) 和 `**` (幂运算)。故正确答案是 **A**。

63. What is the output of the following code: **B**

```
my_list = ["Hello", "Python"]
print("-".join(my_list))
```

A. `HelloPython-`

B. `Hello-Python`

C. `-HelloPython`

详细解答

这道题考察的是 Python 字符串对象的方法 **`join()`**。它是处理字符串序列合并最常用的方式之一。

- **方法语法 `separator.join(iterable)`：**
  - **`separator`** (分隔符)：即代码中的 `"-"`，它将被插入到序列的元素之间。
  - **`iterable`** (可迭代对象)：即代码中的 `my_list`，其内部元素必须是字符串。

逻辑拆解分析：

(a) **理解连接规则：**`join()` 方法会将分隔符插入到列表元素**之间**，而不是每个元素的末尾或开头。

(b) **拼接过程：**

- 列表元素 1：`"Hello"`
- 插入分隔符：`"-"`
- 列表元素 2：`"Python"`
- 最终结果 → `"Hello-Python"`

选项分析：

A. **HelloPython-**：错误。分隔符不会出现在结果字符串的末尾。

B. **Hello-Python**：**正确**。分隔符完美地连接了两个单词。

C. **-HelloPython**：错误。分隔符不会出现在结果字符串的开头。

结论： `join()` 方法的核心逻辑是“将分隔符放置在元素之间”。如果列表中只有一个元素，则不会出现分隔符。故正确答案是 **B**。

64. Please select all correct ways to empty the following dictionary: **C**

```
student = {
    "name": "Emma",
    "class": 9,
    "marks": 75
}
```

A) `del student`
B) `del student[0:2]`
C) `student.clear()`

详细解答

这道题考察的是 Python 字典 (Dictionary) 的管理操作，重点在于区分“删除整个字典对象”与“清空字典内部成员”的不同方法。

- **目标：**使字典变为空字典 `{}`，但保留 `student` 这个变量名。

选项逻辑分析：

(a) **选项 A (`del student`)：**

- **效果：**从内存中删除整个变量名 `student`。
- **后果：**执行后，如果再次尝试访问 `student`，会抛出 `NameError`。这属于“销毁”而非“清空”。

(b) **选项 B (`del student[0:2]`)：**

- **效果：**会引发 `TypeError`。
- **原因：**字典是**无序的映射类型**（Mapping），不支持类似列表那样的切片（Slicing）操作。

(c) **选项 C (`student.clear()`)：**

- **效果：**移除字典中所有的键值对（Key-Value pairs）。
- **状态：**执行后，变量 `student` 依然存在，但其内容变为 `{}`。这是清空字典的标准方法。

结论：只有 `clear()` 方法能在保留变量引用的前提下，安全且完整地移除字典内的所有数据。故正确答案是 **C**。

65. Dictionary keys must be immutable: **A**

A) True

B) False

详细解答

这道题考察的是 Python 字典 (Dictionary) 对 **键 (Key)** 的基本约束。这是理解 Python 映射类型底层逻辑的基础。

- **核心规则：**字典的键必须是 **可哈希的 (Hashable)**。
- **不可变性 (Immutability)：**通常情况下，不可变对象（如字符串、数字、元组）都是可哈希的，因此可以作为键。

逻辑拆解分析：

(a) **为什么要不可变？**字典在底层是基于 **哈希表 (Hash Table)** 实现的。当你存储一个键值对时，Python 会计算键的哈希值来决定存储位置。

如果键是可变的（例如列表），那么在其内容改变后，哈希值也会随之改变，导致无法再找到原来存储的值。

(b) **哪些可以作为键？**

- **合法 (不可变)：** `str`, `int`, `float`, `tuple` (且元组内元素也必须不可变), `bool`。
- **非法 (可变)：** `list`, `dict`, `set`。

选项分析：

A. **True：正确**。在 Python 的标准实现中，为了保证哈希值的稳定性，字典的键必须是不可变类型。

B. **False：**错误。如果键是可变的，会引发 `TypeError: unhashable type` 异常。

结论：字典键的不可变性是确保数据检索效率和准确性的关键。故正确答案是 **A**。

66. In Python, Dictionaries are immutable: **A**

A) False
B) True

详细解答

这道题考察的是对 Python **可变对象 (Mutable)** 与 **不可变对象 (Immutable)** 的分类能力。

- **核心定义：**可变对象是指在创建后，其内容可以被原地（in-place）修改，而无需创建新对象。

逻辑拆解分析：

(a) **字典的特性：**字典（Dictionary）是典型的**可变数据类型**。我们可以随时对字典执行以下操作：

- **添加：**增加新的键值对。
- **修改：**更改已有键所对应的值。
- **删除：**移除现有的键值对。

这些操作都会直接修改原始的字典对象，而不会改变其在内存中的地址（`id`）。

(b) **易混淆点拨：**很多初学者会受到“字典的**键**必须不可变”这一规则的影响，误以为字典本身也是不可变的。

- **键 (Keys)：**必须不可变（如字符串、数字）。
- **字典本体 (Dictionary itself)：**是可变的。

Python 数据类型分类表：

不可变 (Immutable)	可变 (Mutable)
数字 (int, float)	**字典 (dict)**
字符串 (str)	列表 (list)
元组 (tuple)	集合 (set)

结论：由于字典允许增删改操作，它属于可变类型。题目陈述“Dictionaries are immutable”是错误的。故正确答案是 **A**。

67. Select the all correct way to remove the key marks from a dictionary: **AB**

```
student = {
    "name": "Emma",
    "class": 9,
    "marks": 75
}
```

A) `student.pop("marks")`
B) `del student["marks"]`
C) `student.remove("marks")`
D) `student.popitem("marks")`

详细解答

这道题考察的是 Python 字典中删除特定 **键 (Key)** 的多种方法及其语法区别。

- **目标：**从字典 `student` 中精准移除键为 `"marks"` 的键值对。

选项逻辑分析：

(a) **选项 A (`student.pop("marks")`)：**

- **机制：**该方法会移除指定的键并返回其对应的值（在此例中返回 `75`）。
- **结果：**键 `"marks"` 被成功删除。这是最常用的方法，因为它还能捕获被删除的数据。

(b) **选项 B (`del student["marks"]`)：**

- **机制：**使用 Python 的通用删除关键字 `del` 定位到字典的具体键。
- **结果：**直接从内存中删除该键值对。如果键不存在，会抛出 `KeyError`。

(c) **选项 C (`student.remove("marks")`)：**

- **错误分析：**`remove()` 是 **列表 (List)** 的方法，用于删除特定的“值”。字典对象没有 `remove()` 方法，执行会报错。

(d) **选项 D (`student.popitem("marks")`)：**

- **错误分析：**`popitem()` 不需要参数。在 Python 3.7+ 中，它默认删除并返回字典中的 **最后一个**键值对（LIFO）。传入参数会导致 `TypeError`。

总结：

方法	适用对象	特点
`pop(k)`	字典	删除指定键并返回值
`del d[k]`	字典/列表	通用删除指令
`remove(v)`	列表	字典不可用
`popitem()`	字典	无参数，删除最后一项

结论：选项 A 和 B 均能正确且有效地移除指定的键。故正确答案是 **AB**。

68. Select the correct way to access the value of a history subject **A**

```
sampleDict = {
    "class":{
        "student":{
```

```
            "name":"Mike",
            "marks":{
                "physics":70,
                "history":80
            }
        }
    }
}
```

A) `sampleDict['class']['student']['marks']['history']`

B) `sampleDict['class']['student']['marks'][1]`

C) `sampleDict['class'][0]['marks']['history']`

详细解答

这道题考察的是 Python 中 **嵌套字典 (Nested Dictionary)** 的多级索引访问。在处理复杂的数据结构（如 JSON）时，这种逐层剥开（Peeling）的方法非常关键。

- **核心原则：**字典是基于 **键 (Key)** 存储的映射类型。每一层括号 `[]` 内部都必须填入对应的键名。

逻辑路径分析：

(a) **第一层：**`sampleDict['class']` → 进入类字典，得到包含 `student` 的对象。

(b) **第二层：**`...['student']` → 进入学生字典，得到包含 `name` 和 `marks` 的对象。

(c) **第三层：**`...['marks']` → 进入成绩字典，得到包含具体的学科成绩。

(d) **第四层：**`...['history']` → 最终定位到历史学科的值：`80`。

选项分析：

A. **完全正确：**路径中的每一个键名都与字典结构精确对应。

B. **错误：**字典是无序的，不能通过整数索引（如 `[1]`）来访问其中的条目，除非字典的键本身就是整数。

C. **错误：**在 `'class'` 键对应的值是一个字典，而不是列表，因此不能使用索引 `[0]`。

结论：访问嵌套字典必须遵循“层层递进，键名匹配”的原则。故正确答案是 **A**。

69. Select correct ways to create an empty dictionary **AB**

A) `sampleDict = {}`
B) `sampleDict = dict()`
C) `sampleDict = dict{}`

70. Select correct ways to create an empty dictionary **AB**

A) `sampleDict = {}`
B) `sampleDict = dict()`
C) `sampleDict = dict{}`

详细解答

这道题考察的是 Python 中初始化 **空字典 (Empty Dictionary)** 的标准语法。Python 提供了两种主要方式来完成这一操作。

- **方式一：字面量语法 (Literal Syntax)：**使用大括号 `{}`。这是最常用且效率最高的方式。
- **方式二：内置构造函数 (Constructor)：**使用 `dict()` 函数。

选项逻辑分析：

(a) **选项 A (`sampleDict = {}`)：**

- **解析：**在 Python 中，一对空的大括号默认创建一个空字典。
- **注意：**虽然集合 (Set) 也使用大括号，但空集合必须使用 `set()` 来创建，不能使用 `{}`，因为字典在 Python 语言设计中拥有大括号的“优先权”。

(b) **选项 B (`sampleDict = dict()`)：**

- **解析：**调用内置的 `dict` 类构造器，不传递任何参数时，会返回一个新的空字典对象。

(c) **选项 C (`sampleDict = dict{}`)：**

- **解析：语法错误**。在 Python 中，函数或类的调用必须使用圆括号 `()`。将类名与大括号直接连用是不符合语法规则的。

知识对比：

数据类型	字面量方式	构造函数方式
字典 (dict)	`d = {}`	`d = dict()`
集合 (set)	(不可用)	`s = set()`
列表 (list)	`l = []`	`l = list()`

结论：选项 A 和 B 均是 Python 官方认可的创建空字典的正确方式。故正确答案是 **AB**。

71. Select the correct ways to get the value of marks key: **BD**

```
Student = {
    "name": "Emma",
    "class": 9,
    "marks": 75
}
```

A) `m = student.get(2)`
B) `m = student.get('marks')`
C) `m = student[2])`
D) `m = student['marks']`

详细解答

这道题考察的是在 Python 字典中获取特定键对应数值的两种标准操作。字典是映射类型，必须通过 **键 (Key)** 来定位值，而不能使用索引位置。

- **直接访问法：**使用方括号 `dict[key]`。
- **方法访问法：**使用内置函数 `dict.get(key)`。

选项逻辑分析：

(a) **选项 A 和 C (`get(2)` / `student[2]`)：**

- **解析：**这两个选项试图通过整数 2 来获取值。

- **错误原因：**字典是无序的，不支持位置索引。在本字典中，并没有一个键的名字是整数 2，因此 `get(2)` 会返回 `None`，而 `student[2]` 会直接抛出 `KeyError`。

(b) **选项 B (`student.get('marks')`)：**

- **解析：**`get()` 方法接收一个键作为参数。
- **优点：**这是更安全的方式。如果键 `'marks'` 不存在，程序不会报错，而是返回 `None`（或自定义的默认值）。

(c) **选项 D (`student['marks']`)：**

- **解析：**这是获取字典值最直接、最常用的语法。
- **注意：**使用这种方式时，必须确保键 `'marks'` 确实存在于字典中，否则程序会中断并报 `KeyError`。

对比总结：

方式	语法	若键不存在的结果
下标访问	`student['marks']`	抛出异常 (KeyError)
get 方法	`student.get('marks')`	返回 None (不报错)

结论：选项 B 和 D 均使用了正确的键名来提取数值。故正确答案是 **BD**。

72. What is the output of the following dictionary operation **A**

```
dict1 = {"name": "Mike", "salary": 8000}
temp = dict1.pop("age")
print(temp)
```

A) `KeyError: 'age'`

B) `None`

详细解答

这道题考察的是 Python 字典 **`pop()`** 方法在面对不存在的键（Key）时的行为逻辑。

- **`pop(key[, default])` 的运行规则：**
  - 如果 `key` 在字典中，则移除并返回该键对应的值。

– 如果 key 不在字典中,且没有提供 default(默认值)参数,Python 将抛出错误。

逻辑拆解分析:

(a) **检查字典内容:** dict1 中仅包含两个键: "name" 和 "salary"。

(b) **执行操作:** 代码尝试执行 dict1.pop("age")。由于 "age" 这个键在字典中并不存在,且开发者没有在 pop() 方法中提供备选的默认值(例如 pop("age", None))。

(c) **程序反应:** Python 解释器无法完成删除任务,于是立即引发一个 **KeyError**。

对比实验(避坑指南):

- **情况 1(报错):** dict1.pop("age") → **KeyError**。
- **情况 2(安全):** dict1.pop("age", "Not Found") → 返回 "Not Found",程序继续运行。
- **情况 3(get 方法):** dict1.get("age") → 返回 None(注意: get 默认不报错,而 pop 默认报错)。

结论: 由于题目代码未处理键不存在的情况,会导致程序崩溃并报出键错误。故正确答案是 **A**。

73. What is the output of the following code **A**

```
dict1 = {"key1":1, "key2":2}
dict2 = {"key2":2, "key1":1}
print(dict1 == dict2)
```

A) True

B) False

详细解答

这道题考察的是 Python 中字典的 **等值比较 (Equality Comparison)** 逻辑。

- **核心原则：**在 Python 中，两个字典被判定为相等 (==)，当且仅当它们包含 **完全相同的键值对**。
- **顺序无关性：**字典是基于映射（Mapping）的结构。尽管在 Python 3.7+ 中字典会保持插入顺序，但在进行 == 运算时，**键值对出现的先后顺序并不影响相等性的判断**。

逻辑拆解分析：

(a) **检查内容：**

- `dict1` 包含：`'key1': 1` 和 `'key2': 2`。
- `dict2` 包含：`'key1': 1` 和 `'key2': 2`。

(b) **对比过程：**Python 解释器会核对两个字典的长度是否一致，以及 `dict1` 中的每一个键是否都在 `dict2` 中存在且对应的值相等。

(c) **结论：**虽然两个字典在定义时的键值对顺序不同，但它们拥有的“成员”是完全一致的。

对比实验：

- **字典对比：**`{'a':1, 'b':2} == {'b':2, 'a':1}` 结果为 **`True`**。
- **列表对比：**`[1, 2] == [2, 1]` 结果为 **`False`**（列表是有序序列，顺序必须一致）。

结论：只要两个字典的键值对内容一致，等值比较就会返回 `True`。故正确答案是 **A**。

74. Items are accessed by their position in a dictionary and All the keys in a dictionary must be of the same type. **B**

A) True
B) False

```
# For example:
dict1 = {"key1":1, "key2":2, 5: "test"}
```

详细解答

这道题包含两个关于 Python 字典的描述，这两个描述均是 **错误**的。

- **断言 1：通过位置访问 (Accessed by position)**
  - **错误原因：**字典是 **映射类型 (Mapping)**，而不是序列类型（如列表或元组）。在字典中，数据是通过 **键 (Key)** 来检索的，而不是通过数值索引（位置）来访问。
- **断言 2：键必须是相同类型 (Keys must be of the same type)**
  - **错误原因：**字典的键只要是 **不可变 (Immutable)** 且 **可哈希 (Hashable)** 的即可。在同一个字典中，可以混合使用不同类型的键（例如字符串和整数）。

实例代码分析：

```
dict1 = {"key1": 1, "key2": 2, 5: "test"}
```

- 在这个例子中，键既有字符串 (`"key1"`)，也有整数 (`5`)。
- 访问 `"test"` 必须使用 `dict1[5]`，而不能使用位置索引（如 `dict1[2]`，除非字典里恰好有个键是 `2`）。

归纳总结：

特征	描述	结论
访问方式	通过“键”映射	并非通过位置
键的类型	只要可哈希即可	允许混合类型
值的类型	任何类型均可	允许混合类型

结论：由于题目中的两个陈述均不符合 Python 字典的特性，该命题为假。故正确答案是 **B**。

75. Select the correct way to print Emma's age. **B**

```
student = {1: {'name': 'Emma', 'age': '27', 'sex': 'Female'},
           2: {'name': 'Mike', 'age': '22', 'sex': 'Male'}}
```

A) `student[0][1]`

B) `student[1]["age"]`

C) `student[0]["age"]`

详细解答

这道题考察的是嵌套字典的访问，特别是在字典的 **键 (Key)** 为整数时的识别能力。

- **关键陷阱：**在字典中，数字 `1` 是一个具体的 **键**，而不是代表“第二个位置”的索引。
- **数据结构：**外层字典有两个键，分别是整数 `1` 和 `2`。

逻辑拆解分析：

(a) **定位 Emma 所在的子字典：**通过观察发现，Emma 的信息存储在键为 `1` 的值中。

- `student[1]` → `{'name': 'Emma', 'age': '27', 'sex': 'Female'}`

(b) **定位具体的年龄：**在获取到的子字典中，使用键 `"age"` 获取数值。

- `student[1]["age"]` → `'27'`

选项分析：

A. **`student[0][1]`：**错误。字典中没有键为 `0` 的条目，且子字典中也没有键为 `1` 的条目。

B. **`student[1]["age"]`：正确**。外层通过整数键 `1` 访问，内层通过字符串键 `"age"` 访问。

C. **`student[0]["age"]`：**错误。虽然内层键对了，但外层键 `0` 并不存在（Python 字典不会像列表那样自动从 0 开始索引）。

结论：字典访问必须严格匹配定义的键名。由于 Emma 对应的键是 `1`，故正确答案是 **B**。

76. Select all correct ways to copy a dictionary in Python: **A B**

A) `dict2 = dict1.copy()`

B) `dict2 = dict(dict1)`

C) `dict2 = dict1`

详细解答

这道题考察的是 Python 中字典的 **拷贝 (Copying)** 机制。我们需要区分什么是“创建副本”，什么是“创建引用”。

- **目标：**创建一个新的字典对象，使得对新字典的修改（在非嵌套层级）不会影响原字典。

选项逻辑分析：

(a) **选项 A (`dict1.copy()`)：**

- **解析：**这是字典内置的浅拷贝方法。它会创建一个新的字典对象，并填充原字典中的内容。
- **结果：**成功拷贝。

(b) **选项 B (`dict(dict1)`)：**

- **解析：**使用字典构造函数 `dict()`。当你把一个现有的字典作为参数传递给它时，它会遍历该字典并创建一个包含相同键值对的新字典。
- **结果：**成功拷贝。

(c) **选项 C (`dict2 = dict1`)：**

- **解析：**这仅仅是 **引用赋值 (Reference Assignment)**。
- **风险：**此时 `dict1` 和 `dict2` 指向内存中同一个对象。如果你修改了 `dict2`，`dict1` 也会随之改变。这不属于真正意义上的“拷贝”副本。

引用 vs 拷贝示意：

操作	是否创建新对象	修改副作用
`dict2 = dict1`	否 (引用)	修改一个，另一个也变
`dict1.copy()`	是 (浅拷贝)	修改顶层元素互不影响
`dict(dict1)`	是 (浅拷贝)	修改顶层元素互不影响

结论：只有 A 和 B 能够产生独立的新字典对象。故正确答案是 **AB**。

77. What is the output of the following: **C**

```
sampleDict = dict([
    ('first', 1),
    ('second', 2),
    ('third', 3)
])
print(sampleDict)
```

A) `[ ('first', 100), ('second', 200), ('third', 300) ]`

B) `Options: SyntaxError: invalid syntax`

C) `{'first': 1, 'second': 2, 'third': 3}`

详细解答

这道题考察的是 Python 字典构造函数 **`dict()`** 的一种高级用法：通过“键值对序列”来创建字典。

- **构造机制：** `dict()` 可以接受一个可迭代对象（如列表），该对象内部的每个元素必须也是一个包含两个元素的可迭代对象（如元组）。
- **对应关系：** 内部元组的第一个元素 `'first'` 会成为 **键 (Key)**，第二个元素 `1` 会成为 **值 (Value)**。

逻辑拆解分析：

(a) **输入结构：** 输入是一个列表 `[...]`，里面包含了三个元组：`('first', 1)`、`('second', 2)` 和 `('third', 3)`。

(b) **转换过程：**

- `('first', 1)` → `'first': 1`
- `('second', 2)` → `'second': 2`
- `('third', 3)` → `'third': 3`

(c) **最终输出：** Python 会将这些键值对包裹在大括号 `{}` 中，形成标准的字典格式。

选项分析：

A. **错误：** 这依然是一个列表格式，且数值被无故扩大了 100 倍。

B. **错误：**这是完全合法的 Python 语法，不会产生 `SyntaxError`。

C. **`{'first': 1, 'second': 2, 'third': 3}`：正确。**这是列表转字典后的标准字面量表示形式。

结论：通过 `dict()` 构造器处理双元素序列是创建字典的常用方法。故正确答案是 **C**。

78. What is the output of the following dictionary operation: **B**

```
dict1 = {"name": "Mike", "salary": 8000}
temp = dict1.get("age")
print(temp)
```

A) `KeyError: 'age'`

B) `None`

Explanation: The `get` method will not report an error

详细解答

这道题考察的是 Python 字典 **`get()`** 方法的容错机制。这是开发者在不确定某个键是否存在时，首选的取值方式。

- **`get(key[, default])` 的运行规则：**
  - 如果 `key` 存在于字典中，返回其对应的值。
  - 如果 `key` 不存在，且未指定 `default` 参数，则返回 **`None`**。
  - **核心特点：**无论键是否存在，该方法永远不会抛出 `KeyError` 异常。

逻辑拆解分析：

(a) **检查字典内容：**当前 `dict1` 只有 `"name"` 和 `"salary"` 两个键。

(b) **执行检索：**调用 `dict1.get("age")`。由于 `"age"` 不在字典中，且代码没有提供第二个参数（如 `dict1.get("age", 0)`），Python 默认返回其内置的空值对象 `None`。

(c) **打印输出：**变量 `temp` 被赋值为 `None`，因此打印结果为 `None`。

vs dict.get(key)]

对比总结：

访问方式	若键"age" 不存在	结果类型
`dict1["age"]`	抛出 `KeyError`	异常中断
`dict1.pop("age")`	抛出 `KeyError`	异常中断
`dict1.get("age")`	**返回 `None`**	**程序继续运行**

结论： `get()` 方法的设计初衷就是为了提供一种“平稳退化”的访问机制。故正确答案是 **B**。

79. You can use the **b** operator to determine whether a key exists in a dictionary.

a. `&`

b. `in`

c.`^`

d. `?`

详细解答

这道题考察的是 Python 中最常用的 **成员运算符 (Membership Operator)**。在处理字典时，我们经常需要先判断某个键是否存在，以避免程序抛出错误。

- **运算符逻辑：** `key in dictionary`。
- **返回值：** 如果键（Key）存在于字典中，返回 `True`；否则返回 `False`。

选项逻辑分析：

(a) **选项 a (`&`)：** 这是 **按位与 (Bitwise AND)** 运算符。在 Python 中通常用于整数的位运算或集合（set）的交集运算，不能用于检查字典的键。

(b) **选项 b (`in`)：正确。** 这是 Python 官方指定的成员检测运算符。它不仅适用于字典（检查键是否存在），也适用于列表、元组和字符串（检查元素是否存在）。

(c) **选项 c (`^`)：** 这是 **按位异或 (Bitwise XOR)** 运算符。在集合运算中表示对称差集，不具备成员检测功能。

(d) **选项 d (?)**：这是其他编程语言（如 JavaScript 或 C#）中可能出现的“可选链”或“三元运算符”符号。在 Python 语法中，? 并不是合法的逻辑运算符。

代码示例：

```
student = {"name": "Emma", "age": 25}

# 检查键是否存在
if "name" in student:
    print("Key found!")  # 这行会被执行

if "score" in student:
    print("Score found!")
else:
    print("Key not found!") # 这行会被执行
```

结论： `in` 运算符是 Pythonic 风格中最直接、高效的成员检查方式。故正确答案是 **b**。

80. You use **d** to delete an element from a dictionary.

a. the `remove` method
b. the `erase` method
c. the `delete` method
d. the `del` statement

详细解答

这道题考察的是在 Python 中移除字典特定键值对的标准语法。

- **操作对象：**字典是映射类型，删除操作必须指定要删除的 **键 (Key)**。

选项逻辑分析：

(a) **选项 a (remove 方法)：**

- **解析：**`remove()` 是 **列表 (List)** 和 **集合 (Set)** 的方法，用于删除特定的元素值。
- **结论：**字典对象没有 `remove()` 方法，使用它会引发 `AttributeError`。

(b) **选项 b (erase 方法)：**

- **解析：** `erase` 常见于 C++ 的 STL 容器中。
- **结论：** 在 Python 的标准内置类型（如字典、列表）中，不存在 `erase` 方法。

(c) **选项 c (delete 方法)：**

- **解析：** 这是一个迷惑选项。虽然我们要执行“删除”操作，但 Python 并没有名为 `delete()` 的字典方法。

(d) **选项 d (del 语句)：**

- **解析：正确。** `del` 是 Python 的关键字，用于删除对对象的引用。
- **用法：** `del dictionary[key]`。它可以精准地从字典中移除指定的键及其对应的值。

代码对比示例：

```
my_dict = {"a": 1, "b": 2}

# 正确做法：使用 del 语句
del my_dict["a"]

# 另一种正确做法：使用 pop() 方法（本题未列出）
# my_dict.pop("b")

print(my_dict)  # 输出 {}
```

结论： `del` 是 Python 中通用的删除指令，适用于字典、列表及普通变量。故正确答案是 **d**。

81. The **b** function returns the number of elements in a dictionary:

a. `size()`
b. `len()`
c. `elements()`
d. `count()`

详细解答

这道题考察的是 Python 的内置全局函数。在 Python 中，获取容器类对象（如字典、列表、元组、字符串等）包含的元素数量，使用的是统一的标准接口。

- **功能定义：**对于字典而言，该函数返回的是字典中 **键值对 (Key-Value pairs)** 的总数。

选项逻辑分析：

(a) **选项 a (`size()`)：**

- **解析：** `size()` 常见于 Java 或 C++ 的容器操作中。在 Python 中，只有部分第三方库（如 NumPy 或 Pandas）有类似属性，字典对象本身并不具备此方法。

(b) **选项 b (`len()`)：**

- **解析：正确。** `len` 是 length 的缩写，它是 Python 的内置全局函数。
- **原理：** 调用 `len(d)` 时，实际上是触发了字典内部的 `__len__` 魔术方法。

(c) **选项 c (`elements()`)：**

- **解析：** 在 `collections.Counter` 对象中有一个 `elements()` 方法，但它返回的是一个迭代器，而不是数量。

(d) **选项 d (`count()`)：**

- **解析：** `count()` 通常用于统计某个特定元素在列表或元组中出现的次数，并不用于返回容器的总长度。

代码示例：

```
sample_dict = {"name": "Alice", "age": 20, "city": "
    London"}

# 使用 len() 获取长度
print(len(sample_dict))  # 输出: 3
```

结论： `len()` 是 Python 中获取序列或映射长度的最通用工具。故正确答案是 **b**。

82. You can use **a** to create an empty dictionary.

a. `{}`
b. `()`
c. `[]`
d. `empty()`

详细解答

这道题考察的是 Python 中不同内置数据类型的 **字面量 (Literal)** 定义符。

- **核心知识：** 在 Python 中，大括号 `{}` 是字典 (Dictionary) 和集合 (Set) 的共同符号。但在处理 **空容器**时，`{}` 被默认分配给字典。

选项符号对比：

(a) **选项 a (`{}`)：**

- **定义对象：空字典**。
- **注意点：** 若要创建空集合，必须使用 `set()`。

(b) **选项 b (`()`)：**

- **定义对象：空元组 (Tuple)**。元组是不可变序列。

(c) **选项 c (`[]`)：**

- **定义对象：空列表 (List)**。列表是可变序列。

(d) **选项 d (`empty()`)：**

- **解析：** 在 Python 标准库中没有名为 `empty()` 的内置函数。这可能是某些数学库（如 NumPy 的 `np.empty()`）中的函数，但与字典创建无关。

总结表：

符号	对应数据类型	示例
{}	**字典 (dict)**	`{'key': 'value'}`
`[]`	列表 (list)	`[1, 2, 3]`
`()`	元组 (tuple)	`(1, 2, 3)`
`{1, 2}`	集合 (set)	`set()` (空集合写法)

结论：大括号是字典的标志性符号。故正确答案是 **a**。

83. The **a** method returns the value associated with a specified key and removes that key-value pair from the dictionary.

    a. `pop()`
    b. `random()`
    c. `popitem()`
    d. `rand_pop()`

84. The **d** dictionary method returns the value associated with a specified key. If the key is not found, it returns a default value.

    a. `pop()`
    b. `key()`
    c. `value()`
    d. `get()`

85. The **c** method returns all of a dictionary's keys and their associated values as a sequence of tuples.

    a. `keys_values()`
    b. `values()`
    c. `items()`
    d. `get()`

86. What does NumPy stand for? **A**

    A. Numerical Python
    B. New Python
    C. NumPy

87. What is the primary object type in NumPy called? **A**

    A. Array
    B. List
    C. Dictionary

88. What NumPy function is used to create an array? **A**

    A. `array()`
    B. `create_array()`
    C. `make_array()`

89. What type of operations can be performed on NumPy arrays? **C**

    A. Element-wise operations
    B. Matrix operations
    C. Both A and B

90. What NumPy function is used to get the shape of an array? **A**

    A. `shape()`
    B. `get_shape()`
    C. `array_shape()`

91. What NumPy function is used to stack arrays along a new axis? **A**

    A. `stack()`
    B. `combine()`
    C. `join()`

92. What NumPy function is used to add two arrays element-wise? **A**

    A. `add()`
    B. `plus()`
    C. `sum()`

93. What NumPy function is used to multiply two arrays element-wise? **A**

    A. `multiply()`
    B. `times()`
    C. `product()`

94. What NumPy function is used to find the maximum value of an array? **A**

    A. `max()`
    B. `highest()`
    C. `maximum()`

95. What NumPy function is used to find the minimum value of an array? **A**

    A. `min()`
    B. `lowest()`
    C. `minimum()`

96. Which NumPy method is used to create an array from a list? **A**

    A) `array()`
    B) `fromlist()`
    C) `list_to_array()`

97. What NumPy method returns a new array with the elements along an axis? **AB**

    A) `sum()`
    B) `mean()`
    C) `axis()`

```
import numpy as np
x = np.arange(12)
x = x.reshape((3,4))

# np.mean(x, axis=1) returns [1.5, 5.5, 9.5]
# np.sum(x, axis=1) returns [6, 22, 38]
```

98. What NumPy method is used to stack arrays in sequence vertically? **A**

    A) `vstack()`
    B) `hstack()`
    C) `stack()`

99. What NumPy method returns the indices of the maximum values along an axis? **A**

    A) `argmax()`
    B) `where_max()`
    C) `max_indices()`

100. What NumPy method is used to repeat the elements of an array? **A**

    A) `repeat()`
    B) `replicate()`
    C) `copy()`

101. What NumPy method returns the standard deviation of the array elements? **A**

    A) `std()`
    B) `deviation()`
    C) `variance()`

102. What NumPy method returns a new array with one axis removed? **A**

    A) `squeeze()`
    B) `axis_remove()`
    C) `flatten()`

103. What NumPy method returns the cumulative sum of the elements along an axis? **B**

    A) `cumprod()`
    B) `cumsum()`
    C) `cumulative()`

104. What NumPy method returns the indices that would sort an array? **B**

    A) `sort_indices()`
    B) `argsort()`
    C) `index_sort()`

105. What NumPy method returns a view of the array with the same data in memory? **A**

    A) `view()`
    B) `copy_view()`
    C) `same_memory()`

106. What does matplotlib stand for? **C**

    A) Matrices and Plots Library
    B) Markup Language Plotting Tools
    C) Modular Python Library for Plotting

107. What is the primary object used to create plots in matplotlib? **B**

    A) Figure
    B) Axes
    C) Plot

108. What function is used to create a basic plot? **A**

    A) `plot()`
    B) `plt()`
    C) `graph()`

109. What is the most common way to add data to a plot in matplotlib? **B**

A) `add_data()`

B) `plot()`

C) `scatter()`

110. What is the main class used to represent the figure area in matplotlib? **A**

A) `Figure`

B) `Axes`

C) `Canvas`

111. What function is used to display a plot? **A**

A) `show()`

B) `display()`

C) `render()`

112. What function is used to save a figure as an image file? **A**

A) `savefig()`

B) `export()`

C) `save_image()`

113. What property controls the line color in a plot? **A**

A) `color`

B) `c`

C) `linecolor`

114. What property controls the line style in a plot? **A**

A) `linestyle`

B) `style`

C) `line`

115. What property controls the marker in a scatter plot? **A**

A) `marker`

B) `mark`

C) `scattermarker`

116. What function is used to add a title to a plot? **B**

A) `set_title()`

B) `title()`

C) `add_title()`

117. What function is used to add axis labels? **B**

    A) `label()`
    B) `xlabel(), ylabel()`
    C) `axes_labels()`

118. What function is used to add a legend? **A**

    A) `legend()`
    B) `add_legend()`
    C) `key()`

119. What function is used to set the x-axis range? **A**

    A) `xlim()`
    B) `xrange()`
    C) `set_xlimits()`

120. What function is used to set the y-axis range? **A**

    A) `ylim()`
    B) `yrange()`
    C) `set_ylimits()`

121. What object represents the plotting area in a figure? **B**

    A) `Figure`
    B) `Axes`
    C) `Subplot`

122. What function is used to add text annotations to a plot? **B**

    A) `text()`
    B) `annotate()`
    C) `label()`

123. What object is used to customize the font properties in a plot? **A**

    A) `FontProperties`
    B) `Font`
    C) `TextProperties`

124. What function is used to create subplots in a grid layout? **B**

    A) `subplot()`
    B) `subplots()`
    C) `gridplot()`

125. What object is used to customize the figure size and resolution? **A**

    A) `Figure`
    B) `FigureSize`
    C) `FigureCanvas`

126. What is the primary data structure in pandas for storing tabular data? **B**

    A) `Series`
    B) `DataFrame`
    C) `Array`

127. Which pandas method is used to read a CSV file into a DataFrame? **A**

    A) `read_csv()`
    B) `load_csv()`
    C) `import_csv()`

128. How do you select a single column from a DataFrame in pandas? **B**

    A) `df.select_column('column_name')`
    B) `df['column_name']`
    C) `df.get_column('column_name')`

129. What pandas method is used to calculate the mean of a column in a DataFrame? **A**

    A) `mean()`
    B) `average()`
    C) `calculate_mean()`

130. How do you drop a column from a DataFrame in pandas? **C**

    A) `df.drop_column('column_name')`
    B) `df.remove_column('column_name')`
    C) `df.drop('column_name', axis=1)`

131. What pandas method is used to fill missing values in a DataFrame? **C**

    A) `fill_missing()`
    B) `fill_nan()`
    C) `fillna()`

132. How do you sort a DataFrame by a specific column in pandas? **B**

A) `df.sort('column_name')`

B) `df.sort_values('column_name')`

C) `df.order_by('column_name')`

133. What pandas method is used to group data in a DataFrame? **C**

A) `group_by()`

B) `group()`

C) `groupby()`

134. How do you rename a column in a DataFrame in pandas? **B**

A) `df.rename_column('old_name', 'new_name')`

B) `df.rename(columns={'old_name': 'new_name'})`

C) `df.rename('old_name', 'new_name')`

135. What pandas method is used to save a DataFrame to a CSV file? **C**

A) `save_csv()`

B) `write_csv()`

C) `to_csv()`

136. How do you select multiple columns from a DataFrame in pandas? **B**

A) `df.select_columns(['col1', 'col2'])`

B) `df[['col1', 'col2']]`

C) `df.get_columns(['col1', 'col2'])`

137. What pandas method is used to calculate the median of a column in a DataFrame? **A**

A) `median()`

B) `middle()`

C) `calculate_median()`

138. How do you filter rows in a DataFrame based on a condition in pandas? **B**

A) `df.filter(condition)`

B) `df[condition]`

C) `df.apply(condition)`

139. What pandas method is used to drop rows with missing values in a DataFrame? **C**

A) `drop_missing()`

B) `remove_nan()`

C) `dropna()`

140. How do you reset the index of a DataFrame in pandas? **A**

    A) `df.reset_index()`
    B) `df.set_index()`
    C) `df.reindex()`

141. What pandas method is used to merge two DataFrames based on a common column? **B**

    A) join()
    B) merge()
    C) combine()

142. How do you calculate the sum of a column in a DataFrame in pandas? **C**

    A) df.sum('column_name')
    B) df.calculate_sum('column_name')
    C) df['column_name'].sum()

143. What pandas method is used to calculate descriptive statistics of a DataFrame? **A**

    A) describe()
    B) summary()
    C) stats()

144. How do you drop duplicate rows from a DataFrame in pandas?

    A) df.drop_duplicates()
    B) df.remove_duplicates()
    C) df.drop_duplicates(inplace=True)

145. What pandas method is used to pivot a DataFrame based on column values? **A**

    A) pivot_table()
    B) pivot()
    C) reshape()

146. Create a 1D NumPy array with values from 0 to 9.

```
arr1d = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# or
arr1d = np.arange(10)
# or
arr1d = np.linspace(0, 9, 10)
```

147. Create a 2D NumPy array with shape (3, 3) filled with zeros.

```
arr2d = np.zeros((3, 3))
```

148. Create a 3x3 identity matrix using NumPy.

```
identity_matrix = np.eye(3)
```

149. Create a 1D NumPy array with 10 random integers between 1 and 100.

```
random_integers = np.random.randint(1, 100, 10)
```

150. Reshape the array from exercise above into a 2D array with shape (2, 5).

```
reshaped_array = random_integers.reshape(2, 5)
```

151. Multiply each element in the array from exercise above by 2.

```
multiplied_array = reshaped_array * 2
```

152. Calculate the mean of the array from exercise above.

```
mean_value = multiplied_array.mean()
```

153. Find the maximum value in the array from exercise 150.

```
max_value = multiplied_array.max()
```

154. Create a 2D NumPy array with shape (4, 4) filled with random floats between 0 and 1.

```
random_floats = np.random.random((4, 4))
```

155. Transpose the array from exercise above.

```
transposed_array = random_floats.T
```

156. Create a 1D NumPy array with 5 evenly spaced values between 0 and 10.

```
evenly_spaced_array = np.linspace(0, 10, 5)
```

157. Extract all odd numbers from the array from exercise above.

```
odd_numbers = evenly_spaced_array[evenly_spaced_array % 2 !=
   0]
```

158. Create a 2D NumPy array with shape (3, 3) filled with ones.

```
ones_array = np.ones((3, 3))
```

159. Add the array from exercise above to the identity matrix from exercise 147.

```
added_array = ones_array + identity_matrix
```

160. Multiply the array from exercise above by the array from exercise 146.

```
multiplied_array = np.dot(added_array, arr2d)
```

161. Calculate the sum of each column in the array from exercise 159.

```
column_sums = multiplied_array.sum(axis=0)
```

162. Calculate the dot product of the arrays from exercise 146 and exercise 147.

```
dot_product = np.dot(arr2d, identity_matrix)
```

163. Find the indices of the non-zero elements in the array from exercise 1.

```
nonzero_indices = np.nonzero(arr1d)
```

164. Sort the array from exercise 148 in ascending order.

```
sorted_array = np.sort(random_integers)
```

165. Calculate the square root of each element in the array from exercise above.

```
sqrt_array = np.sqrt(sorted_array)
```

166. Create a pandas Series from a list of integers [1, 2, 3, 4, 5].

```
series = pd.Series([1, 2, 3, 4, 5])
```

167. Create a pandas DataFrame from a dictionary with keys 'Name', 'Age', and 'City'.

```
data = {'Name': ['John', 'Jane', 'Mike'],
        'Age': [25, 30, 35],
        'City': ['New York', 'London', 'Paris']}
df = pd.DataFrame(data)
```

168. Read a CSV file named 'data.csv' into a pandas DataFrame.

```
df = pd.read_csv('data.csv')
```

169. Display the first 5 rows of the DataFrame from exercise 167.

```
df.head(5)
```

170. Display the last 3 rows of the DataFrame from exercise 167.

```
df.tail(3)
```

171. Display the column names of the DataFrame from exercise 167.

```
df.columns
```

172. Select the 'Age' column from the DataFrame from exercise 167.

```
df['Age']
```

173. Select the row at index 2 from the DataFrame from exercise 167.

```
df.loc[2]
```

174. Select rows 3 to 7 from the DataFrame from exercise 167.

```
df.loc[3:7]
```

175. Add a new column 'Gender' to the DataFrame from exercise 167 with values ['M', 'F', 'M', 'F', 'M'].

```
df['Gender'] = ['M', 'F', 'M', 'F', 'M']
```

176. Calculate the mean of the 'Age' column in the DataFrame from exercise 167.

```
df['Age'].mean()
```

177. Calculate the maximum value in the 'Age' column in the DataFrame from exercise 167.

```
df['Age'].max()
```

178. Sort the DataFrame from exercise 167 by the 'Age' column in ascending order.

```
df.sort_values('Age')
```

179. Filter the DataFrame from exercise 167 to only include rows where 'Age' is greater than 30.

```
df[df['Age'] > 30]
```

180. Group the DataFrame from exercise 167 by the 'City' column and calculate the mean age for each city.

```
df.groupby('City')['Age'].mean()
```

181. Rename the column 'Age' to 'Years' in the DataFrame from exercise 167.

```
df.rename(columns={'Age': 'Years'}, inplace=True)
```

182. Drop the 'City' column from the DataFrame from exercise 167.

```
df.drop('City', axis=1, inplace=True)
```

183. Save the modified DataFrame from exercise 181 to a new CSV file named 'modified_data.csv'.

```
df.to_csv('modified_data.csv', index=False)
```

184. Merge two DataFrames, df1 and df2, on a common column 'ID'.

```
merged_df = pd.merge(df1, df2, on='ID')
```

185. Perform an inner join between two DataFrames, df1 and df2, on a common column 'ID'.

```
inner_join_df = pd.merge(df1, df2, on='ID', how='inner')
```