Python 计算导论 Cheating Sheet

001. `print()`

- 转义字符: `\n`: 换行 (new line), `\t`: 制表符 (tab)。(在字符串前加 `r` 可以取消转义字符: `r"..."`)
- `""`, `"""""""`: 前者不自动换行，需要用 `\n` 换行；后者自动换行，需要用 `\` 取消换行。
- f-string: Formatted string literals (格式化字符串字面量)。

002. `ord()`: 转化字符为 ASCII 码。

003. `chr()`: 转化 ASCII 码为字符。

004. **运算符**: `+ - * /`: 加减乘除; `//`: 整除; `%`: 求余 (模); `**`: 乘方。

005. `type(n)`: 返回 n 的类型。

006. `id(n)`: 返回 n 的内存地址。

007. `int()`, `float()`, `str()`: 类型转换。

008. `eval("...")`: 识别字符串，计算表达式。

009. `bool()`: 当且仅当空容器、0、`False`、`None` 时返回 `False`，否则返回 `True`。

010. `range(start=0, end, step=1)`: 输出从 start 到 end 的以 step 为公差的等差数列，**不包括 end**，产物是迭代器。

011. **循环控制**:

- `break`: 循环终止，执行循环后的语句。
- `continue`: 本次循环结束，继续下一个循环。
- `return`: 函数终止。

012. `return a if expre else b`: 如果 expre 满足，输出 a，否则输出 b。

013. `for/while-else`: `else` 后的语句执行当且仅当循环正常结束。

014. **位运算**: `&`: and, `|`: or, `~`: Not, `^`: XOR (异或), `>>`: 右移, `<<`: 左移。

015. `global` 关键字: 函数内部变量声明为全局变量，调用并修改外部变量。

016. `*args` 和 `**kwargs`: 接受任意数量的变量，前者包装为 `tuple`，后者包装为 `dictionary`。注意*和**是解包操作。

017. `list(a)`: a 如果是 string，那么每个字符为列表的一个元素；a 如果是迭代器，那么自动生成列表。

018. `list.index(value, start=0, end=-1)`: 搜索索引，end 不被包括在内。默认全局，返回第一个索引。

019. `list.count(x)`: 返回 x 出现的次数。

020. `list.clear()`: 删除所有元素；`list.pop(index)`: 根据索引删除 (并返回该删除的值)。但是用法是 `del list[index]`。

021. `list.remove(value)`: 删除某个数值，但是不返回；`del item`: 删除 item。

022. `list.insert(index, value)`: 在某个位置插入 value，index 是插入后的索引位置。一定注意是插入后的位置。

023. `list.extend(list')`: 在 list 后面加入 list'，改变 list；`list + list'` 返回新的列表。这点和字符串类似。

024. `list.sort()`: 同类型可以排序；`list.reverse()`: 翻转列表。

025. `list[i:j]`: 切片，依旧包含 i 不包含 j，生成新的列表；`list[a:]`: 从 a 切到最后；`list[:a]`: 从开始切到 a-1。

026. `list[i:j:k]`: i 开始，j-1 结束，k 是步长。切片一定不会超限。

027. `list[:]`: 复制列表；`list.copy()`: 浅复制，只复制一层；`copy.deepcopy(list)`: 全部复制 (需 `import copy`)。

028. **列表推导**: `[expression for target in iterable if condition]`。

029. `for x in list`: 无法获取 index；`for i in range(len(list))`: 无法直接获取值。用`enumerate`更好。

030. `for i, j in enumerate(list)`: i 是索引，j 是索引对应的值。

031. `tuple`: 是不可修改的。但是 tuple 内部的元素倘若是可修改的（如列表），则可以修改该元素的内容。

032. `dictionary = {key_1: value_1, ...}`: 字典定义。

033. **Key 的限制**: 字典的关键字必须是不可改变的 (immutable)。`list`, `dictionary` 不可以做 key，但是 `tuple` 可以。

034. **字典访问**: `dict[key]`；`dict[key] = new_value`: 修改或者重新赋值 (取决于 key 是否存在)。

035. `get(key, default)`: 查 key，存在返回对应值，否则返回 default 值。不存在**不插入** `{key: default}`。

036. `setdefault(key, default)`: 除了会在不存在的时候插入这对值，别的都和 `get` 一样。显然用 get 方法更安全。

037. `dict.popitem()`: 删除最后一个插入的 key 和 value。

038. `dictionary`: 是可修改的，可以复制，类似 list。

039. **字典融合**: `z = {**x, **y}`。如果 key 重复，后者覆盖前者。

040. `dict.keys()`: 返回所有的键为一个 iterable，需自行转 list 或 tuple。

041. `dict.values()`: 返回所有的值为一个 iterable。

042. `dict.items()`: 返回所有的 (key, value) 对为一个 iterable。

043. `str.split(mark=" ")`: 根据 mark 分裂字符串成列表，默认为空格。

044. `str.count()`: string 也具有 count 的功能，可以在一定范围内寻找。

045. `str.find(sub, start, end)`: 在范围内找 sub 的第一个 index，end 不被包括在内。返回 -1 意味着不存在。

046. `import module`: 调用 module 的函数的时候需要用 `module.func()`。

047. `from module import func`: 从顶层开始搜索，直到文件夹为止。

048. **时间复杂度 (List/Tuple/Str)**: 查找、插入、删除: $O(n)$；访问、末尾加入: $O(1)$。但是插入在开头是 $O(n)$。

049. **时间复杂度 (Dict/Set)**: 运用哈希表，空间复杂度高，但是上述操作对字典和集合都是 $O(1)$。

050. **错误类型 (Exceptions)**:

`SyntaxError`	语法错误，比如括号不封闭。
`IndentationError`	缩进错误，该对齐的没对齐。
`NameError`	变量没找到，大概率是没定义。
`AttributeError`	属性错误，大概率类型搞错了或对象没有该属性。
`TypeError`	类型错误，瞎操作导致的。
`ValueError`	数值错误，值不在容器中或参数无效。
`IndexError`	索引错误，索引超限。

`KeyError`	键值错误，字典中不存在该键。
`ZeroDivisionError`	数学错误，以 0 为除数。

051. `random.randint(a, b)`: 返回一个随机整数 N，满足 $a \leq N \leq b$。

052. `random.randrange(start, stop, step)`: 返回一个整数，但是 stop 不包括在范围内。

053. `random.random()`: 返回一个 0 到 1 之间的随机浮点数。

054. `random.uniform(a, b)`: 返回一个 a, b 之间的随机浮点数。

055. `random.seed(seed=sys_time)`: 根据种子生成一个随机浮点数。不指定默认是系统时间。

056. `random.choice(iterables)`: 从容器中随机选一个出来。

057. `random.choices(iterables, weights, cum_weights, size)`: 从容器中随机一个 size 的子列出来。

058. `random.shuffle(iter)`: 打乱；`random.sample(iter, k)`: 随机选长度为 k 的子容器出来。

059. sys 模块:

- `sys.version`: Python 版本。
- `sys.argv`: 获取传给 Python 的命令行参数。`argv[0]`: 文件名, `argv[1]`: 第一个参数...
- `sys.path`: Python 搜索的路径列表。
- `sys.exit([arg])`: 退出程序。
- `sys.maxsize`: 变量能容纳的最大整数。
- `sys.stdin`: 标准输入流；`sys.stdout`: 标准输出流；`sys.stderr`: 标准错误流。

060. os 模块:

- `os.name`: 返回操作系统名称 (nt: Windows, posix: Linux)。
- `os.environ`: 读取环境变量。
- `os.chdir()`: 改变工作目录，等价于在命令行执行 cd 命令。
- `os.mkdir()`: 创建单个目录，只能创建一级，存在会报错。
- `os.rename()`: 重命名。若已经存在，在 win 环境下会失败，Linux 会覆盖。
- `os.remove()`: 删除文件 (删除文件夹用 `os.rmdir()`)。

- `os.startfile()`: 启动程序运行，启动关联的默认程序。
- `os.walk()`: 遍历目录树，产生三元组 `(dirpath, dirnames, filenames)`。

061. **open()**: `obj = open(file_name, mode, buffering)` (名字, 模式, 缓冲策略)

- `r`: 只读，文件必须存在否则报错。
- `w`: 只写，会覆盖文件，不存在则创建。
- `a`: 在文件末尾追加，不存在则创建。
- `x`: 独占创建，必须不存在，否则报错。
- `*b`: 加在 rwax 后面，二进制模式，用来处理非文本文件。
- `*t`: 文本模式，默认模式，通常省略。
- `*+`: 可读写。

062. `obj.closed`: 判断文件是否关闭；`obj.name`: 返回文件名；`obj.mode`: 文件读写模式。

063. `obj.write(string)`: 写入字符串到文件中,不会自动换行。`obj.writelines(list)`: 注意没有 `\n` 不会自动换行。

064. `read(n=None)`: 读取 n 个字节，返回一个字符串，默认全部返回。

065. `readline()`: 逐行读取，只读取一行，保留 `\n`，返回字符串。

066. `readlines()`: 一次性读取所有行，返回每一行字符串组成的列表，保留 `\n`。

067. `with open(name, mode) as obj`: 缩进块内操作，运行结束自动关闭，安全。

068. `for line in obj`: file 对象是可迭代的。

069. `iter(obj)`: obj 可以为元组、列表等可迭代的东西，这个函数可以获取它的迭代器。

070. `next(iter)`: 对于一个迭代器，每次读下一个迭代的内容。

071. **itertools 模块**:

- `product(p, q, ...)`: 返回笛卡尔积。
- `permutations(p, r)`: 返回一个 r 长度的所有可能排序，无重复元素的元组。
- `combinations(p, r)`: 返回一个 r 长度的有序的、无重复元素的元组。

072. `callable()`: 对于一个函数 `test = func()`，`callable(test)` 返回 True 当且仅当 func 可以被调用。

073. `@classmethod`: 标记静态方法 (类方法)。

074. `@decorator_name`: 装饰器，输入函数为变量的方法。

075. `@property`: 性质，用于标记方法为属性。

076. **海象运算符**: 允许开发者在表达式内部进行变量赋值: `x := 1 + 1`。

077. **Lambda 匿名函数**: `lambda var_1, var_2: expression`，用于在表达式内部处理。

078. `sorted(iterable, key=func)`: sorted 高阶用法，通过比较 func(item) 的值来排序。

079. **List 高阶用法**: lambda 函数也可以是一个字典的 value。

080. `map(func, iterable)`: 对 iterable 的每个元素使用 func，输出同款 iterable (在 Python 3 中是迭代器)。

081. `filter(func, iterable)`: 遍历 iterable 中的每个元素，若 `func(i) == True`，输出到新容器中。

082. `nonlocal`: `global` 是直接打通到最外层，`nonlocal` 只打通到上一层嵌套函数。注意只有在修改的时候才需要这俩。

083. **LEGB**: 查询顺序: 局部作用域 (Local) → 外层函数作用域 (Enclosing) → 全局作用域 (Global) → 内置作用域 (Built-in)。