1 cf's 4A problem

2020-01-12 09:19:39.244631

divide a even ingeter into two parts, each of them is even

这说明了这个整数能被 2 整除（因为 even + even = even)，且不能为 2.

2 Python 的编码问题

2020-01-12 12:17:27.624951

尝试用 `print('...', file=out_put_file)` 来进行文本输出，发现输出的文本不是用 `UTF-8` 来编码的，而是好像用的国标（国家的标准?）

在 `str` 中字符是用 `unicode` 来编码的，是没有被 `encode` 的二进制数据，输出到文件，输出的内容编码是由系统的配置决定的。

猜想：`open` 时会指定 `encoding` 可以解决这个问题。

`open` 时指定 `encoding` 会有带有编码信息的属性的文本对象。因为有状态的影响，所以该对象的 `write method` 会用编码下的二进制实现去写文件。

而 `print` 可以指定 `file` ，但是它（应该）不会读取文件内部的编码状态，（这个时候我想起来了，有时用 `u8` 编码的时候好像会在文件头留下一个特别的标记?）而是直接用系统指定的编码格式去输出二进制数据去写数据。

猜想：`print` 是把文件输出到 `sys.stdout` 上（应该，至少意思是这样）。输出的环境是由系统的配置/环境决定的，为了环境能够正常显示文本数据，就应该会用配置下的文件编码格式来编码 `Unicode` 的数据流，这也是为什么它不会去读取文本文件的编码状态的原因吧?（甚至还没有参数可以调节输出的编码状态，这样）

3 matplotlib 的字体输出

2020-01-12 12:19:30.764825

这个问题我之前也记在白皮书上了的，但是在写一点也 ok。

matplotlib 以什么字体输出是根据 cont.family，font.serif，font.sans-serif 等决定的。（text.usetex = True 是指排版用 TEX，字体用配置指定的，还是全部由 TEX 自己决定的?）

font.family 指定输出的字体的字系，有诸如 serif 可以选择。

一般 serif，中文字体感觉不错的有经典的 SimSun （宋体），而 sans-serif，用于编程的 Consolas 也不错。

关于修改配置，可以在文件中修改，或者用 plt.rcParams['xxxx.yyyy'] = data 来修改的。（当然可能有其他的修改输出格式的办法）

在 axes.texta 中，用 family 参数可以临时指定字系。应该也可能有其他的方法可以。

4 matplotlib 的 cmap

2020-01-12 12:20:10.551049

之前网上的示例用到了‘ RdYlGn ’的 cmap，通过 plt.get_cmap 来获取的。之前以为 RdYlGn 的中间是白的，以为它好像是：$Red \to Orange \to White \to Lightgreen \to Green$ 的样子。结果看了才知道并不是这样子的，它中间反而是黄的，是 Yl 的样子，有一说一，确实，这也太白了一点。

也有颜色表，比如 tab20c，一格一格的。

不是很确定，返回的 cmap 对象可能调用一个元素为 int 或者 float 的可迭代对象，然后返回对应的颜色列表。

5 使用 TikZ 创建条件图

2020-01-12 12:20:53.074716

我最近非常想要设计一个语法统一优美的作图语言，我简直被 python 宠坏了。

唉.....

先提一句: `xtikzxfill[orange](1ex, 1ex)circle (1ex);` 可以画个圆。(这个 TikZ 太麻烦了。

而 `xtikz xdraw[->] begin -- end;` 可以用来画箭头。

今天在知乎上面逛了一圈，有人说可以用 `python` 来搞个 UML 图，试试就试试。

p.s. 我现在用的是 `graphviz` 的 `dot` 语言。还可以吧，但是总是有点麻烦。语法格式也很不统一。

6 LaTeX 中的对齐问题

2020-01-12 12:21:33.898432

LaTeX 中有两种对齐方式：(1) 环境对齐，(2) 命令对齐。其中环境的话是上下都空了个间隔，用命令会好一点。但怎么说呢，命令的话会作用到一个 `xpar` 。

注：VsCode 中用 `$nunber` 来表示捕获的分组。

7 Python 中的 setup.py

2020-01-12 12:22:21.429750

通过 setup.py 文件，可以使用命令 `python setup.py install` 来安装包。(注，要在 setup.py 目录下运行才会 ok，也就是说，`cwd` 是 setup.py 的父目录)

也可以使用 `setuptools.setup(...)` 的那啥来指定外部名，之后就可以在外面直接用了。

8 python 的 pyyaml 问题

2020-01-12 12:22:55.901702

直接用 pyyaml 会有点问题。现在我是在用 ruamel.yaml 的第三方包。

效果不错，但是还有要改进的地方。我想去参与开发这个项目。有一说一，这个项目还不错的说。

9 cf's 71A problem

2020-01-12 09:50:00.711916

在 C++ 中因为对象没有特殊方法（当然，构造方法和解构方法除外，还有运算符重载，如果这也算的话）

所以说并不是所有对象都可以转化为字符串的。从另一个方面来讲，如果过于底层的话，的确不需要用到这种方法（比如 Python 的 `__str__` 方法）。

（话说，我是在说服自己吗??）

对于基本的数据，`std::string` 提供了特殊的方法，在 std 的命名空间里，提供了方法 `std::to_string(number_type)` ，

另外，如果输出要换行的话，也不能忘掉。

p.s. 在上一题中，我也要时刻注意输入数据的范围。

10 cf's 118A problem

2020-01-12 15:35:41.865311

遇到了几个有意思的问题。

首先时如何把字符串转换为小写的字符串。在 python 中只需要使用 `str.lower()` 就可以得到一个拷贝了。（谢天谢地，我现在特别怀念 python）但是很明显这在 C++ 中时行不通的。

第一种转换方法是使用 `transfrom` 函数再配上一个 lambda 函数。transfrom 是由标准库 algorithm 提供的，在关于它的介绍网站说的那样（在 https://en.cppreference.com/w/cpp/algorithm/transform 中），它的几个声明之一是：

```
template< class InputIt, class OutputIt, class
    UnaryOperation >
OutputIt transform( InputIt first1, InputIt last1,
    OutputIt d_first,
                UnaryOperation unary_op );
```

它会把 unary_op 作用到这些 $[frist1, last1)$ 上去,而输出到 $[frist2, +\infty)$ 上去。

而 lambda 函数，最简单的形式是这两种：

```
[ captures ] ( params ) { body }
[ captures ] { body }
```

回到主题来,这个函数就是为了封装住来自 cctype 的函数 `std::tolower` 的函数原型是 `int tolower(int ch);`

以上是第一种方法。

第二种方法是使用 boost 库，不表。

之后遇到了有多个使用或和等于的逻辑判断符，换个思路，其实用 set 可能也是一个不错的思路。有一点很有意思，如果找不到，一般会返回该容器的 `.end()` 的值。

11 cf's 85A problem

2020-01-13 01:32:53.936011

和上次那道关于字符串的题很像。都涉及到了把字符串转换为相应的小写形式。

关于转换的函数，应该是下列的样子：

```
std::string lower(std::string str){
    std::transform(str.begin(), str.end(), str.begin(),
            [](unsigned char c){return std::tolower(c)};
    return str;
}
```

我自己在使用中的时候，lambda 函数没有加上 `return` 语句，下次一定。（lambda 函数是一个黑盒，必须要有输入有输出）。

12 关于 C++ 中的 using 和 typedef

2020-01-13 17:25:19.221487

总是因为泛型的原因要声明很长的变量类型，但是其实有为变量类型设置别名的方法。

第一点是使用 `typedef` 来重命名变量。格式如下：

```
    typedef org new;
```

另外的方法是使用 `using` 来进行命名变量的工作。这是 C++11 起才开始支持的，语法会更好一些，更统一化一些。另外它还支持模板操作。普通的用法是这样：

```
    using new = org;
```

或者说这样：

```
template <typename T> using my_type = whatever<T>;

my_type<int> my_var;
```

这样相较而言，typedef 就好像是宏定义的一样（当然并不是）。

13 Python 中的 ___new___ 方法

2020-01-14 09:28:06.576783

在 Python 中的 `__init__` 方法一般只是用来设置属性用的。换言之，`__init__` 只是在使用 `__new__` 后获得对象后给对象加属性而使用的特殊方法。

所以说，真正可以获得对象的方法，还是要用到 `__new__` 的特殊方法。

而一般的类设计是不需要定义 `__new__` 特殊方法的，原因是对于它们来说继承属于 `object` 类的 `__new__` 方法就 OK 了（事实上是没有 object 这个类型的声明的，但是关于这个概念，每个设计类的人都要理解，因为它是用 Python 的解释器实现的，是一切类的基石，就像内建类型一样，不过话说，object 也的确是内建类型）。

所以可以认为 `__new__` 是特殊方法中的特殊方法。是调用类之后的之后第一个被调用的类方法。而它生成的对象更是其他方法的基础。

因为这个，`__new__` 不同与其他方法一样，反之，它被传入的第一个参数是 `cls`，是类对象，而不像其他方法一样传入的是实例对象，也就是 `self`。在最后的最后，`__new__` 会返回一个类对象所对应的实例对象。

从类到对象，一般而言只需要调用 `object.__new__(cls)` 方法就可以了。如果想对自己的对象加入更多的细节，都可以在自己的类下的 `myclass.__new__` 定义余下的，甚至还可以实现元类。

当然如果有选择的话，在自己的对象下实现 `__init__` 来定义，这永远是最优选择，就如 Python 之禅所说的一样。当然从另一个方面来看的话，我们会发现 `__init__` 并没有我们想象的那么必不可缺。很多时候甚至可以找到其他的办法来实现 `__init__` 方法所能办到的。但是其他的方法一般来说完全没必要，简单的 `__init__` 已经简单得够招人喜欢了。

在 `__new__` 中，为了使用父类所已经完成了的工作，也可以使用强有力的 `super()`，它的参数还有几个可以传值呢，从而为继承提供了更好的基础。

14 Python 中的 hash

2020-01-15 02:24:11.383948

想要完成的功能：为不同的类型的对象进行一个独一无二的 hash 标记。

为了完成这个功能，我看了看 Python 的标准库，`hashlib` 库。`hashlib` 库为 hash 算法提供了良好的支持，比如说要计算 my-object 的 sha-1 的哈希值，可以这样：

```
m = hashlib.sha1()           # 新建sha1对象
m.update(str(my_object))
```

```
m.digest()                    # sha-1值
m.hexdigest()                 # 表示为16进制字符串的sha-1值
```

(?):update 可以直接接受字符串吗？它不用被编码吗？