第一题：

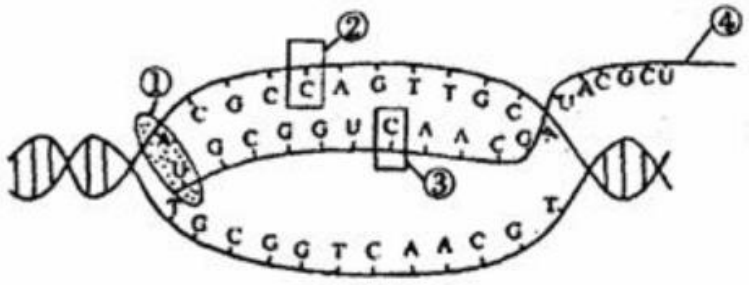
该题主要的目的：根据 20 种氨基酸的密码子表，将给定的 DNA 序列翻译为蛋白质序列。

密码子的第一位（5'端）		密码子的第二位					密码子的第三位（3'端）
		T	C	A	G		
T		TTT: Phe F	TCT: Ser S	TAT: Tyr Y	TGT: Cys C	T	
		TTC: Phe F	TCC: Ser S	TAC: Tyr Y	TGC: Cys C	C	
		TTC: LeT L	TCA: Ser S	TAA: Ter *	TGA: Ter *	A	
		TTC: LeT L	TCG: Ser S	TAG: Ter *	TGG: Trp W	G	
C		CTT: LeT L	CCT: Pro P	CAT: His H	CGT: Arg R	T	
		CTC: LeT L	CCC: Pro P	CAC: His H	CGA: Arg R	C	
		CTA: LeT L	CCA: Pro P	CAA: Gln Q	CGC: Arg R	A	
		CTG: LeT L	CCG: Pro P	CAG: Gln Q	CGG: Arg R	G	
A		ATT: Ile I	ACT: Thr T	AAT: Asn N	AGT: Ser S	T	
		ATC: Ile I	ACC: Thr T	AAC: Asn N	AGC: Ser S	C	
		ATA: Ile I	ACA: Thr T	AAA: Lys K	AGA: Arg R	A	
		ATG: Met M	ACG: Thr T	AAG: Lys K	AGG: Arg R	G	
G		GTT: Val V	GCT: Ala A	GAT: Asp D	GGT: Gly G	T	
		GTC: Val V	GCC: Ala A	GAC: Asp D	GGC: Gly G	C	
		GTA: Val V	GCA: Ala A	GAA: GlT E	GGA: Gly G	A	
		GTG: Val V	GCG: Ala A	GAG: GlT E	GGG: Gly G	G	

该代码根据用户输入的 frame 的值（在 1-6 之间）的不同，主要有两种模式：

（1） frame 为 1，2，3，则直接对给定的序列进行转录翻译。

（2） frame 为 4，5，6，需要将其转换为逆反序列，再进行转录和翻译。



而 frame 这个输入的变量，在这行代码中的作用就是规定翻译的起始位点。

frame	**翻译的链**	**start_position**
1	正链	0
2	正链	1
3	正链	2
4	负链	0
5	负链	1
6	负链	2
<1 or >6	/	显示报错信息

我对这段代码进行了一部分的修改，因为在题目给定的代码中，函数 reverse_complement()和 transcribe()是未定义的。猜测其主要功能分别为将序列转换为逆反序列和转录（即为下图）。

```
    if frame>=4:
        dna = reverse_complement(dna)
        #Startposition des Frames berechnen
        start_pos = frame - 4
    else:
        start_pos = frame - 1
    #wandle DNA in RNA um
    rna = transcribe(dna)
    #Verwende Stepsize 3, beginnend by start_pos
    #bis zum letzten Triplet
```

所以，为了使得该代码能够正常的运转，我引用了 biopython 模块中的两个已知的具有相似功能的函数。

过程如下（这四行是我在源代码的基础上添加的）：

```
#import modules
from Bio.Seq import Seq
dna = Seq(dna) #后面 dna 为函数的输入
rev_dna = dna.reverse_complement()
rna = dna.transcribe()
```

下面展示这四行代码的主要做了什么？

```
(base) [xxzhang@mu02 python_dir]$ python
Python 3.6.13 |Anaconda, Inc.| (default, Feb 23 2021, 21:15:04)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from Bio.Seq import Seq
>>> dna = Seq("ATATAAT")
>>> rev_dna = dna.reverse_complement()
>>> rev_dna
Seq('ATTATAT')
>>> rna = dna.transcribe()
>>> rna
Seq('AUAUAAU')
>>>
```

原先的序列为“ATATAAT”。

当我们对其使用 reverse_complement()函数取逆反后，得到了序列“ATTATAT”。

```
                    A T A T A A T
                    | | | | | | |
取反:               T A T A T T A
```

在反序列的基础上，首尾调换: A T T A T A T

而 transcribe()的作用则是把原先的序列中的 T 变为 U。

```
                    A T A T A A T
                    | | | | | | |
RNA 转录:           A U A U A A U
```

在介绍完上述的基础上，下面正式对该代码进行解释:

```
(base) [xxzhang@mu02 python_dir]$ python code_1.py
CMIDTI*PH
```

对 code_1.py 运行输出结果，与答案一致。

该代码的首先定义了两个变量:

（1） dna 为一个字符型的变量，为我们要进行转换的 dna 序列;

（2） codon_map 为一个字典型的变量，用于存放密码子表的信息。对于字典我这里通俗的解释，就是由无数的键值对构成。键和值之间由“:”来表示映射关系。如 AUA 键对应的值为 I，从生物学意义的角度上看，就是 AUA 密码子编码为氨基酸 I。

```
dna = "ATGCGGTCAGATAGTATCGATCATGCA"
```

```
codon_map = {
    'AUA':'I', 'AUC':'I', 'AUU':'I', 'AUG':'M',
    'ACA':'T', 'ACC':'T', 'ACG':'T', 'ACU':'T',
    'AAC':'N', 'AAU':'N', 'AAA':'K', 'AAG':'K',
    'AGC':'S', 'AGU':'S', 'AGA':'R', 'AGG':'R',
    'CUA':'L', 'CUC':'L', 'CUG':'L', 'CUU':'L',
    'CCA':'P', 'CCC':'P', 'CCG':'P', 'CCU':'P',
    'CAC':'H', 'CAU':'H', 'CAA':'Q', 'CAG':'Q',
    'CGA':'R', 'CGC':'R', 'CGG':'R', 'CGU':'R',
    'GUA':'V', 'GUC':'V', 'GUG':'V', 'GUU':'V',
    'GCA':'A', 'GCC':'A', 'GCG':'A', 'GCU':'A',
    'GAC':'D', 'GAU':'D', 'GAA':'E', 'GAG':'E',
    'GGA':'G', 'GGC':'G', 'GGG':'G', 'GGU':'G',
    'UCA':'S', 'UCC':'S', 'UCG':'S', 'UCU':'S',
    'UUC':'F', 'UUU':'F', 'UUA':'L', 'UUG':'L',
    'UAC':'Y', 'UAU':'Y', 'UAA':'*', 'UAG':'*',
    'UGC':'C', 'UGU':'C', 'UGA':'*', 'UGG':'W',
}
```

接着就是本代码的主体部分，作者定义了一个函数，名为 translate()。在 python 中对于函数的声明，需要用关键词“def”，该函数的主要的输入变量为两个，一为我们名为 dna 的变量，第二个为名为 frame 的变量，默认值为 1（当用户不规定 frame 的值时，frame 取 1）。

注：这两个变量是仅在函数内发挥作用的局部变量，与咱们之前提到的 dna 的那个变量无关。

```
def translate(dna, frame=1):
```

在函数中，首先作者定义了一个存放输出氨基酸序列的字符变量 aa_seq：

```
    aa_seq = ""
```

字符变量可以通过“+”的方式进行拼接，如下：

```
>>> a= "Good!"
>>> b = "Idea!"
>>> a+b
'Good!Idea!'
```

接下来这一段 if……else……语句是一种判断结构：if 后面跟的是要满足的条件。

```
    if frame < 1 or frame > 6:
        print("[ERROR]: Frame must be between 1 and 6")
        return None
    #Check if Readingframe >=4 and
    #reverse-complement dna
    if frame>=4:
        dna = reverse_complement(dna)
        #Startposition des Frames berechnen
        start_pos = frame - 4
    else:
        start_pos = frame - 1
```

计算机在执行代码的时候，是逐行的进行的，所以，首先，它会先判断 frame 的值是否在 1-6 之间，如果不在，就是第一个 if 语句，那么程序会输出报错信息：“ERROR……”，该程序返回“None”，由于程序已经有一个返回值，则接下来的语句便不会再执行。而当输入的 frame 在 1-6 之间，则第一个 if 语句的判断就不满足了，程序会跳过这段代码，执行下面的语句。

第二个 if 语句，由 if……else 组成，意识就是说，如果满足（frame=4,5,6），则执行 if 下面的语句，否则(frame=1,2,3)，则执行 else 下面的语句。(注：关于 frame 的生物学意义前面那张表已经介绍，此不赘述) 在这段 if/else 语句中规定了是否对序列取逆反，以及我们在翻译的时候的起始位点的值。

接下来这一段，是这个函数的核心内容。

```
    #wandle DNA in RNA um
    rna = transcribe(dna)
    #Verwende Stepsize 3, beginnend by start_pos
    #bis zum letzten Triplet
    for position in range(start_pos, len(rna)-2, 3):
        codon = rna[position:position+3]
        if codon in codon_map:
            aa_seq += codon_map[codon]
     return aa_seq
```

首先，作者使用 transcribe()函数，将候选的 dna 序列转换为 rna 序列（T➔U）。然后，接下来就是一个 for 循环，大致的过程就是对这段序列由前面规定的起始位点，以 3 为单位，逐步提取密码子，然后再将这个密码子，与之前定义的字典

中的键进行比较，如果字典中存在这个键，则把这个键所对应的值取出来，然后把该值逐个拼接成输出的氨基酸序列。最后函数输入得到的该氨基酸序列。

下面对 for 循环进行详细的介绍：

（1） range（start_pos,len(rna)-2,3）

range 函数可以将其理解为在一定的范围内取数。该函数的前两个数，规定了起始和终止的数，最后一个数，规定了步长。

注：len()函数指的是求 rna 这个字符串的长度，所以 len(rna)-2 从本质上来讲是一个数值。

```
>>> len("AAAATT")
6
>>> len("AAAzzxTxxaxs")
12
>>> len("AAAzzxTxxaxxxzxzs")
17
```

以下是一些取值的范例：

Start_pos	Len(rna)-2	步长	取值范围
0	8	3	0 3 6
1	8	3	1 4 7
2	8	3	2 5 8
3	8	3	3 6

注：在计算机中，索引是从 0 开始的，而不是从 1 开始。所以，如果想要表示变量的第一个值，则其索引为 0。

```
for position in range(start_pos, len(rna)-2, 3):
```

所以上述这个代码，它的意思是，在 start_pos 到 rna 的长度-2 的这样的一个数值区间内，以 3 为步长的范围内进行取值，position 表示的就是每一次的取值，如 0，3，6 等。

```
codon = rna[position:position+3]
```

Rna 是一个字符串，所以这样代码的意思就是说，取字符串[position,position+2]

这一段的字符串，将其赋值为 codon（注意是 2,不是 3）。

```
>>> rna = "AUAAUGCC"
>>> rna[0:3]
'AUA'
>>> rna[3:6]
'AUG'
>>>
```

取出这一段 RNA 序列之后，我们肯定想知道，这一段密码子对应的是那一个氨基酸。因为我们前面规定了密码子和氨基酸对应关系的字典，所以，我们在这里只需要判断，我们取的这一段序列是否是字典中已知的“键”。就是下面这个 if 语句，如果该段是字典中已知的键，则取出相对应的值，并将该值，赋值给 aa_seq 这个变量。

```
if codon in codon_map:
    aa_seq += codon_map[codon]
```

最后，当 for 循环在规定的值的范围内，遍历完成之后，循环结束。此时，aa_seq 就保留了 rna 序列上密码子所对应的氨基酸的序列。最后，函数将其作为输出。

```
return aa_seq
```

所以，translate（）函数到这里就定义完成了。

用户只需要,向该函数内扔任意 dna 和 frame 的值,即可得到对应的输出的结果。

以下是一些测试的结果:

```
>>> a = translate("ATGCGGTCAGATAGTATCGATCATGCA",frame=4)
>>> a
'CMIDTI*PH'
>>> a = translate("ATGCGGTCAGATAGTCA",frame=1)
>>> a
'MRSDS'
>>> a = translate("ATGCGGTCAGATAGTCTTGGA",frame=2)
>>> a
'CGQIVL'
>>> a = translate("ATGCGGTCAGATAGTCTTGGA",frame=5)
>>> a
'PRLSDR'
>>> a = translate("ATGCGGTCAGATAGTCTTGGA")
>>> a
'MRSDSLG'
>>> a = translate("ATGCGGTCAGATAGTCTTGGA",frame=1)
>>> a
'MRSDSLG'
```