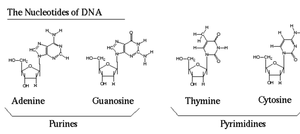
Mount drive

In [1]:
```
#from google.colab import drive
#drive.mount('/content/drive/')
```

Start of Lesson - Validating and counting nucleotides

DNA Toolkit file



In [2]:
```
#  Define the nucleotides that we want to find

Nucleotides = ['A', 'C', 'G', 'T']  # A=Adenine, C=Cytosine, G=Guanosine, T=Thymine
```

Check the sequence to make sure it is a DNA String

In [3]:
```
def validateSeq(dna_seq):
  seq_upper = dna_seq.upper() # Comparing Upper case against Upper case
  for element in seq_upper:
    if element not in Nucleotides:
      return False

  return seq_upper
```

Test the validate sequence function

In [4]:
```
random_dna_str1 = 'AGCTCTCT'
random_dna_str2 = 'AGCTCTCTX'

result1 = validateSeq(random_dna_str1)
result2 = validateSeq(random_dna_str2)

print(f'Result for sequence 1: {result1}')
print(f'Result for sequence 2: {result2}')
```

```
Result for sequence 1: AGCTCTCT
Result for sequence 2: False
```

Creating a random DNA sequence for testing

In [5]:
```
import random

randDNAStr = ''.join([random.choice(Nucleotides) for i in range(80)]) # length can be changed
result3 = validateSeq(randDNAStr)
print(f'Result for random sequence:\n{result3}')
```

```
Result for random sequence:
CTCACTAACAGCACGAATTCAGATTCATGCATGCAGGTATCCTAATGGATAATTTGATCGCTATGATAGTTGCGGACTGC
```

Count the frequency of each nucleotide present in the sequence

In [6]:
```
def Count_Freq_Nuc(seq):
    FreqDict = {
        'A': 0,
        'C': 0,
        'G': 0,
        'T': 0
    }

    for nucleotide in seq:
        FreqDict[nucleotide] += 1 # count

    return FreqDict
```

Test Count Frequency function

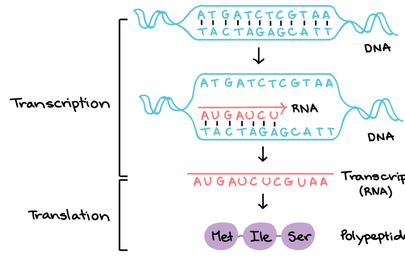
In [7]:
```
Freq_randDNAStr = Count_Freq_Nuc(result3)
print(f'Frequency of Nucleotide in random sequence: {Freq_randDNAStr}')
```

```
Frequency of Nucleotide in random sequence: {'A': 23, 'C': 17, 'G': 17, 'T': 23}
```

End of Lesson - Validating and counting nucleotides

Start of Lesson - Transcription, Reverse Complement

> For this lesson, we are focusing on Transcription, not Translation
>
> Transcription is the first step in gene expression, in which information from a gene is used to construct a functional product such as a protein. The goal of transcription is to make a RNA copy of a gene's DNA sequence. For a protein-coding gene, the RNA copy, or transcript, carries the information needed to build a polypeptide (protein or protein subunit). Eukaryotic transcripts need to go through some processing steps before translation into proteins.



Transcription function

In [8]:
```
def transcription(seq):
  # DNA to RNA Transcription
  return seq.replace('T', 'U') # replace all Thymine nucleotide with U
```

Test transcription function

In [9]:
```
Transcripted_Seq = transcription(result3)
print(Transcripted_Seq)
```

```
CUCACUAACAGCACGAAUUCAGAUUCAUGCAUGCAGGUAUCCUAAUGGAUAAUUUGAUCGCUAUGAUAGUUGCGGACUGC
```

Summary of what we've done so far

In [10]:
```
print(f'Result for random sequence:\n{result3}\n')
print(f'Length of random sequence:\n{len(result3)}\n')
print(f'Frequency of Nucleotide in random sequence:\n{Freq_randDNAStr}\n')
print(f'Transcripted Sequence from random sequence:\n{Transcripted_Seq}')
```

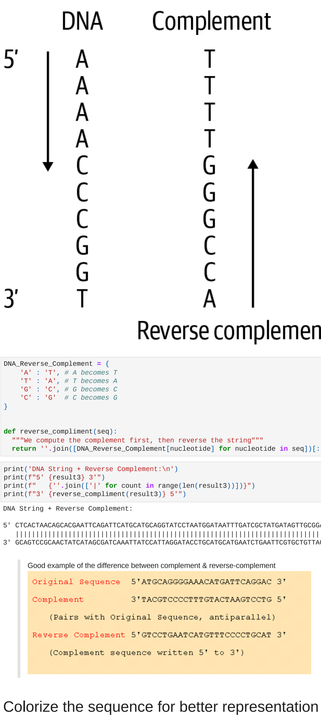
```
Result for random sequence:
CTCACTAACAGCACGAATTCAGATTCATGCATGCAGGTATCCTAATGGATAATTTGATCGCTATGATAGTTGCGGACTGC

Length of random sequence:
80

Frequency of Nucleotide in random sequence:
{'A': 23, 'C': 17, 'G': 17, 'T': 23}

Transcripted Sequence from random sequence:
CUCACUAACAGCACGAAUUCAGAUUCAUGCAUGCAGGUAUCCUAAUGGAUAAUUUGAUCGCUAUGAUAGUUGCGGACUGC
```

Reverse Complement function



In [16]:
```
DNA_Reverse_Complement = {
    'A' : 'T', # A becomes T
    'T' : 'A', # T becomes A
    'G' : 'C', # G becomes C
    'C' : 'G' # C becomes G
}


def reverse_compliment(seq):
  """We compute the complement first, then reverse the string"""
  return ''.join([DNA_Reverse_Complement[nucleotide] for nucleotide in seq])[::-1]
```

In [17]:
```
print('DNA String + Reverse Complement:\n')
print(f"5' {result3} 3'")
print(f"   {''.join(['|' for count in range(len(result3))])}")
print(f"3' {reverse_compliment(result3)} 5'")
```

```
DNA String + Reverse Complement:

5' CTCACTAACAGCACGAATTCAGATTCATGCATGCAGGTATCCTAATGGATAATTTGATCGCTATGATAGTTGCGGACTGC 3'
   ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
3' GCAGTCCGCAACTATCATAGCGATCAAATTATCCATTAGGATACCTGCATGCATGAATCTGAATTCGTGCTGTTAGTGAG 5'
```

> Good example of the difference between complement & reverse-complement
>
> Original Sequence 5'ATGCAGGGGAAACATGATTCAGGAC 3'
>
> Complement 3'TACGTCCCCTTTGTACTAAGTCCTG 5'
>
> (Pairs with Original Sequence, antiparallel)
>
> Reverse Complement 5'GTCCTGAATCATGTTTCCCCTGCAT 3'
>
> (Complement sequence written 5' to 3')

Colorize the sequence for better representation

In [19]:
```
def colored(seq):
    bcolors = {
        'A' : '\033[92m',
        'C' : '\033[94m',
        'G' : '\033[93m',
        'T' : '\033[91m',
        'U' : '\033[91m',
        'reset' : '\033[0;0m'
    }

    tmpStr = ""

    for nuc in seq:
        # if the key matches
        if nuc in bcolors:
            tmpStr += bcolors[nuc] + nuc
        else:
            tmpStr += bcolors['reset'] + nuc

    return tmpStr + '\033[0;0m'
```

In [20]:
```
print(f"5' {colored(result3)} 3'")
print(f"   {''.join(['|' for count in range(len(result3))])}")
print(f"3' {colored(reverse_compliment(result3))} 5'")
```

```
5' CTCACTAACAGCACGAATTCAGATTCATGCATGCAGGTATCCTAATGGATAATTTGATCGCTATGATAGTTGCGGACTGC 3'
   ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
3' GCAGTCCGCAACTATCATAGCGATCAAATTATCCATTAGGATACCTGCATGCATGAATCTGAATTCGTGCTGTTAGTGAG 5'
```

In [ ]:
```

```