



1785

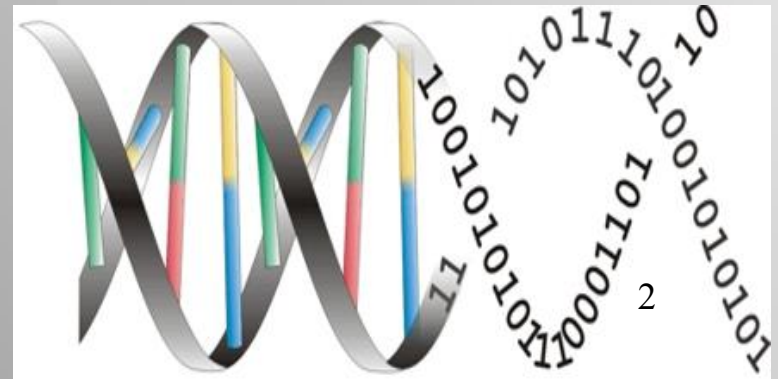
# The University of Georgia



# Programming for Computational & Systems Biology

Instructor: Paul Xie

Tue. & Thr. 9:35~10:50



# Overview of Course

- Introduction to programming skills in computational and systems biology. Topics include **real world examples**, such as **processing** genome or proteome data, and **analyzing** large-scale data. The idea of “big data” will be emphasized to help students with their coding skills to **discovering** new knowledge in biomedical sciences and **solving** biomedical problems.

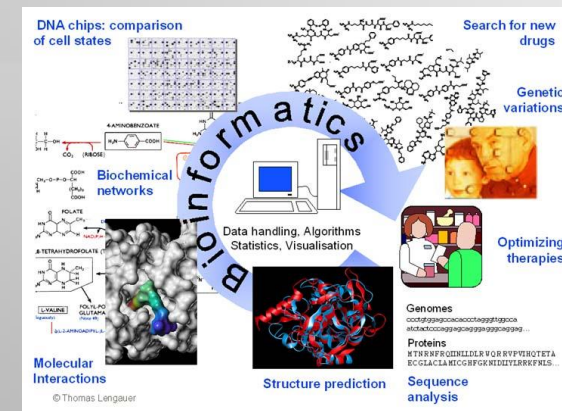
# Course Format

- Computer labs + lectures (3 hours/week)
- Coding concepts (some knowledge about biological data)
- 6-8 assignments, published on eLC, (30-40%)
  - Please upload them by the due days
- Paper review (10-20%)
- 1 term paper (50-60%)

# Topics



- -Omics data, e.g. Genomics, Proteomics...
- Basic Programming, e.g. I/O, variables, string, loop, regular express, array...etc.
- Data retrieval & processing
- Data analysis
- Model & building model
- Clustering and Classification
- Prediction





# Data Array: List & Dictionary Transcription & Translation

Instructor: Paul Xie (4)

Scarites	C	T	T	A	G	A	T	C	G	T	A	C	C	A	A	-	-	-	A	A	T	A	T	T	A	C
Carenum	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	A	-	T	A	C	-	T	T	T	A	C
Pasimachus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	T	A	T	A	A	G	T	T	T	A	C
Pheropsophus	C	T	T	A	G	A	T	C	G	T	T	C	C	A	C	-	-	-	A	C	A	T	A	T	A	C
Brachinus armiger	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	T	C
Brachinus hirsutus	A	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	T	A	T	A	T	A	C
Aptinus	C	T	T	A	G	A	T	C	G	T	A	C	C	A	C	-	-	-	A	C	A	A	T	T	A	C
Pseudomorpha	C	T	T	A	G	A	T	C	G	T	A	C	C	-	-	-	-	-	A	C	-	-	-	-	-	C

# This Week

- Random number
- Data array
  - List
  - Dictionary
- Average & Std. Dev.



# Group Discussion



brainstorm



Sequence alignment of Histone H1 (residues 120-180) across four species: Human, Mouse, Rat, and Chimpanzee. The alignment shows conserved regions (indicated by asterisks) and non-conserved regions (indicated by dots).

Species	Sequence
HUMAN	KKASKPKKAASKAPTCKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASKPKKAKPVK
MOUSE	KKAAKPKKAASKAPSKPKATPVKKAKKKPAATPKKAKKPKVVKPVKASKPKKAKTVK
RAT	KKAAKPKKAASKAPSKPKATPVKKAKKKPAATPKKAKKPKIVKPVKASKPKKAKTVK
COW	KKAAKPKKAASKAPSKPKATPVKKAKKKPAATPKKPKTVKAKPVKASKPKKTKPVK
CHIMP	KKASKPKKAASKAPTCKPKATPVKKAKKKLAATPKKAKKPKTVKAKPVKASKPKKAKPVK

Conservation analysis: Asterisks (\*) indicate conserved amino acids. Dots (.) indicate non-conserved amino acids. The alignment shows high conservation in the first and last regions, with some non-conservation in the middle.

NON-CONSERVED AMINO ACIDS

Conservative

Conservative

Non-conservative

Conservative

Non-conservative

Semi-conservative

Non-conservative

Conservative

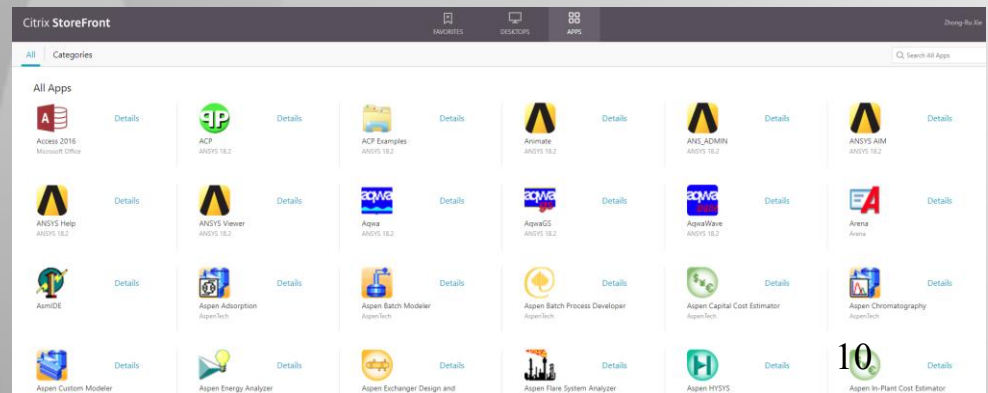
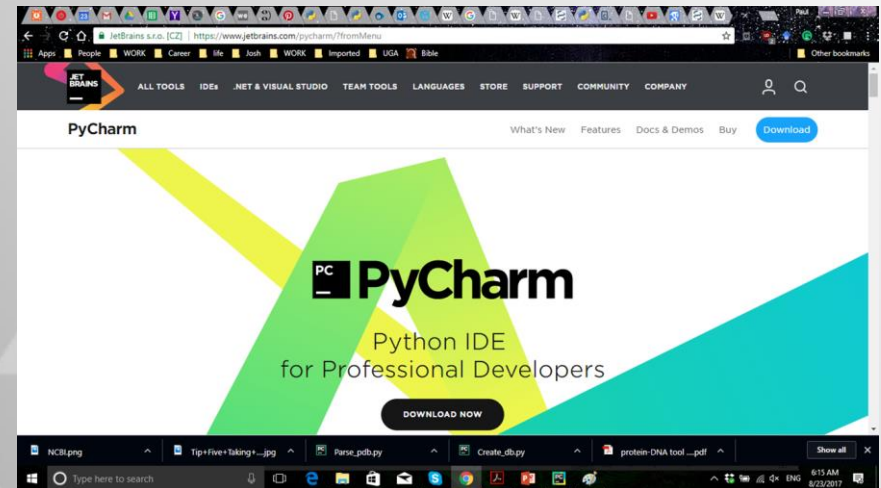


# Python Time



# Open PyCharm

- Go to UGA CENGR Mylab
  - mylab.engr.uga.edu
- Login Citrix
  - Apps
  - PyCharm
  - New project



- Array
  - **List** is a collection which is **ordered** and **changeable**. Allows **duplicate** members. []
  - **Tuple** is a collection which is **ordered** and **unchangeable**. Allows **duplicate** members. ()
  - **Set** is a collection which is **unordered** and **unindexed (unchangeable)**. **No duplicate** members. {}
  - **Dictionary** is a collection which is **unordered**, **changeable** and **indexed**. **No duplicate** members. {key:200}

# Code

```
#array  
def main():  
    cars = ["Ford", "Volvo", "BMW"]  
    for i in cars:  
        print(i)  
  
main()
```

```
#array
def show_array(cars):
    print(len(cars), 'cars.\nThey are:')
    for i in cars:
        print(i)
    for i in range(10):
        print('-', end='')
    print()
```

.....

# Code

```
.....  
def main():  
    cars = ["Ford", "Volvo", "BMW"]  
    show_array(cars)  
    cars.append("Honda") #  
    show_array(cars)  
    cars.pop(2)  
    show_array(cars)  
    cars.insert(1, "Toyota")  
    show_array(cars)  
    print(cars.index("Honda"))  
    cars.remove("Volvo")  
    show_array(cars)  
    cars.sort()  
    show_array(cars)  
    cars.reverse()  
    show_array(cars)  
main()
```

# Exercise

- Random number → array
- Calculate max, min, mean (average)
- Check redundant
  - Double loop
- Sort
  - Bubble sort



# Water Level Marker



- An array  $a[ ]$
- A variable  $x$  to record highest (lowest) value
- (For loop)
  - Compare  $a[i]$  and  $x$
  - Record the highest value



# Average

- An array  $a[n]$
- A variable  $s$  to record summary
- (For loop)
  - $s += a[i];$
- $s /= n;$

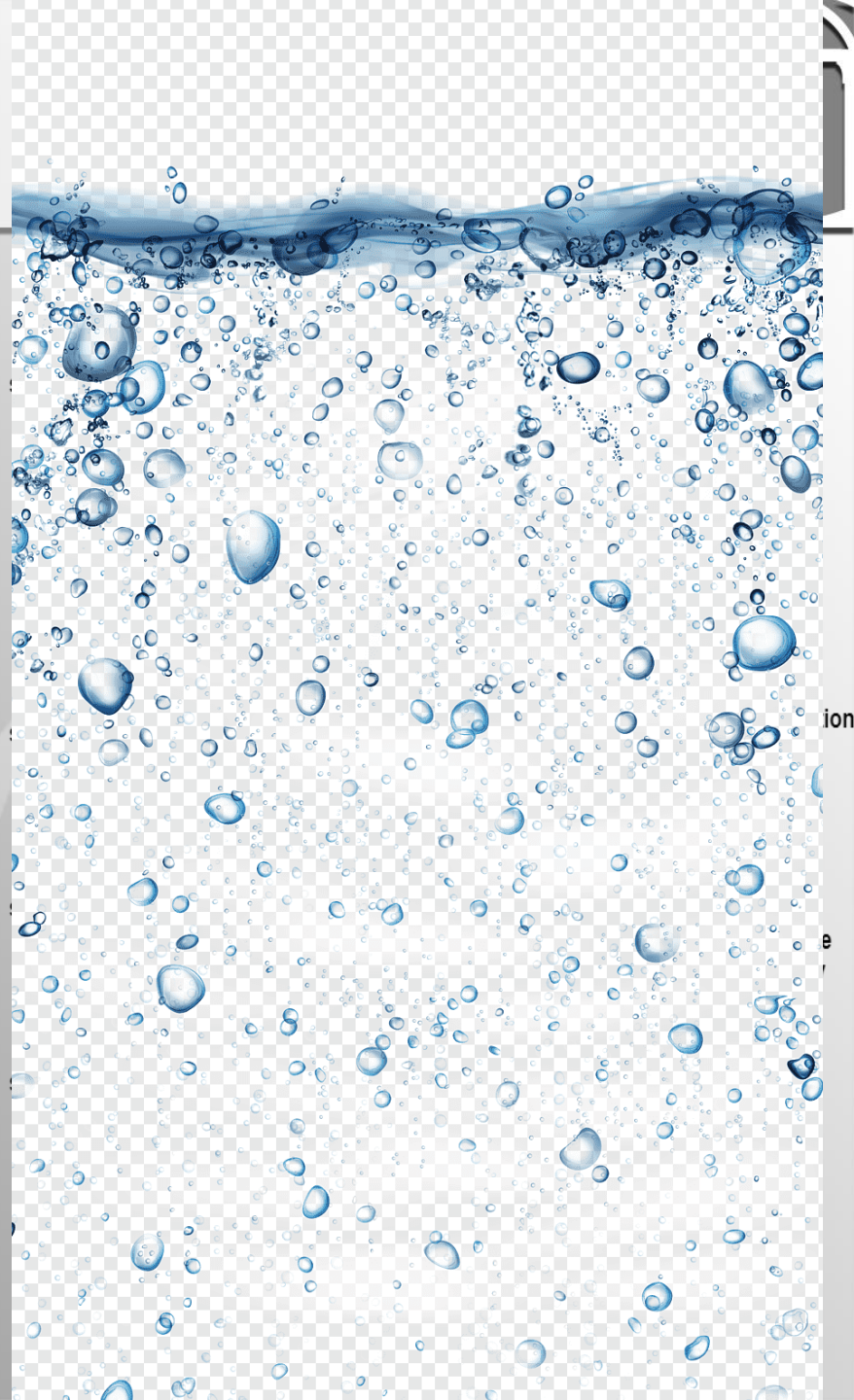


- Average and Standard deviation

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2} = \sqrt{\frac{1}{N} \left( \sum_{i=1}^N x_i^2 \right) - (\bar{x})^2} = \sqrt{\left( \frac{1}{N} \sum_{i=1}^N x_i^2 \right) - \left( \frac{1}{N} \sum_{i=1}^N x_i \right)^2}.$$

# Bubble Sort

- An array  $a[n]$  contains  $n$  numbers
- How to sort them?
- Big bubbles is lighter (go up)

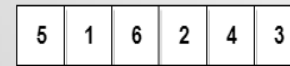


# Bubble Sort

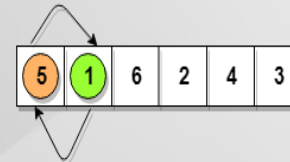


- An array  $a[ ]$
- Double loops (loop #1 and #2)
- Compare numbers and switch

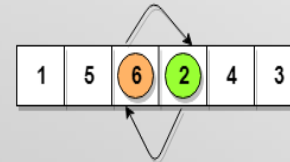
5>1  
so interchange



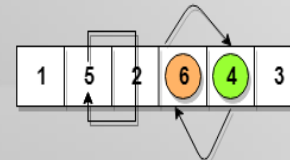
5<6  
No swapping



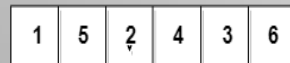
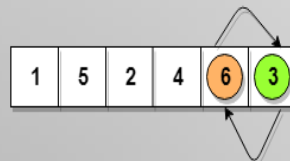
6>2  
so interchange



6>4  
so interchange



6>3  
so interchange



This is first insertion

similarly, after all the iterations, the array gets sorted



# Non-redundant



- Lottery
- 5 or 6 non-redundant balls (numbers)
- For loop
  - Generate ball[i]
  - Compare balls



# Taiwan Lottery



- Prize
  - 1<sup>st</sup> (6 matches) : 82%
  - 2<sup>nd</sup> (5+special): 7%
  - 3<sup>rd</sup> (5 matches): 6.5%
  - 4<sup>th</sup> (4+special): 4.5%
  - 5<sup>th</sup> (4 matches): \$80
  - 6<sup>th</sup> (3+special): \$40
  - 7<sup>th</sup> (2+1 or 3 matches): \$8
- ~56% of incomes





# Exercises

- sort: from big to small
- List all Prime from 1 to 100
  - For loop
  - If /else
  - Array
  - Size unknown
- Store factorial in an array, e.g.  $1! \sim 10!$ 
  - Do we need to calculate  $1!$ ,  $2!$ ,  $3!$ , ... separately?

# Multiplication Table



- Double Loop
- A loop inside another

```
a = []; b = []; c = []
```

```
for i in range(10):
```

```
    k = []; l = []; m = []
```

```
    for j in range(10):
```

```
        k.append(i+1)
```

```
        l.append(j+1)
```

```
        m.append((i+1)*(j+1))
```

```
    a.append(k)
```

```
    b.append(l)
```

```
    c.append(m)
```

MULTIPLICATION TABLE				
1	2	3	4	5
1x1=1	1x2=2	1x3=3	1x4=4	1x5=5
2x1=2	2x2=4	2x3=6	2x4=8	2x5=10
3x1=3	3x2=6	3x3=9	3x4=12	3x5=15
4x1=4	4x2=8	4x3=12	4x4=16	4x5=20
5x1=5	5x2=10	5x3=15	5x4=20	5x5=25
6x1=6	6x2=12	6x3=18	6x4=24	6x5=30
7x1=7	7x2=14	7x3=21	7x4=28	7x5=35
8x1=8	8x2=16	8x3=24	8x4=32	8x5=40
9x1=9	9x2=18	9x3=27	9x4=36	9x5=45
10x1=10	10x2=20	10x3=30	10x4=40	10x5=50
6	7	8	9	10
1x6=6	1x7=7	1x8=8	1x9=9	1x10=10
2x6=12	2x7=14	2x8=16	2x9=18	2x10=20
3x6=18	3x7=21	3x8=24	3x9=27	3x10=30
4x6=24	4x7=28	4x8=32	4x9=36	4x10=40
5x6=30	5x7=35	5x8=40	5x9=45	5x10=50
6x6=36	6x7=42	6x8=48	6x9=54	6x10=60
7x6=42	7x7=49	7x8=56	7x9=63	7x10=70
8x6=48	8x7=56	8x8=64	8x9=72	8x10=80
9x6=54	9x7=63	9x8=72	9x9=81	9x10=90
10x6=60	10x7=70	10x8=80	10x9=90	10x10=100

# 2D Array

- `a = [[0]*m for i in range(m)]`
- Or, `b = [1,2,3,4,5]`, `c = [2,4,6,8,10],...`
- `a.append(b)`
- `a.append(c)`
- ...
- `matrix`

# Exercise

- Create a 2D array (list)
- 2d array for Dawgs score
- 2d array for your assignment scores
- Pascal triangle and Magic square

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1

15	15	15	
8	1	6	15
3	5	7	15
4	9	2	15



Georgia Bulldogs football  
1st in SEC East

GAMES

NEWS

STANDINGS

PLAYERS



3 Georgia

30

Final  
8/31



Vanderbilt

6



Murray State

17

Final  
9/7



Georgia

63



Arkansas State

0

Final  
Sat, 9/14



3 Georgia

55



10 Notre Dame

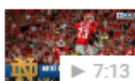
17

Final  
Sat, 9/21



3 Georgia

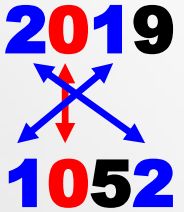
23



# Exercise



- Please design a number guessing game called 1A2B
- Please randomly generate a 4-digit number with 4 different number (for example 2019)
- A player make a first guess, say 1024, compare two number2, 1 correct digit (0) has correct position → 1A; and 2 correct digits has incorrect positions → 2B
- Based on previous hints, keep guess until get it right



# Open a File

- Syntax
- `File_id = open(file_name, operation)`
- `'r','w','a'`
- `File_id.read()`
- `File_id.readline()`
- `File_id.readlines()`

# Syntax

- Array
- Dictionary
  - Key-value



# Code

```
#dictionary  
def main():  
    cars = {  
        "brand": "Toyota",  
        "model": "Corolla",  
        "year": 2003  
    }  
    print(cars)  
    cars["year"] = 2018  
    cars["color"] = "red"  
    cars["owner"] = "paul"  
    cars["seller"] = "Toyota Athens"  
    cars["plate"] = "Georgia"
```

-----

# Code

```
print(len(cars))  
for i in cars:  
    print(i)  
for i in cars.keys():  
    print(i)  
for i in cars.values():  
    print(i)  
for x,y in cars.items():  
    print(x,y)  
cars.popitem()  
cars.pop("seller")  
del cars[]  
print(cars)
```

# Exercise

- Calculate the occurrence & frequency of amino acids (or nucleotides)
  - Open file(s)
  - Dictionaries
  - Loops
  - Counting
  - presentation

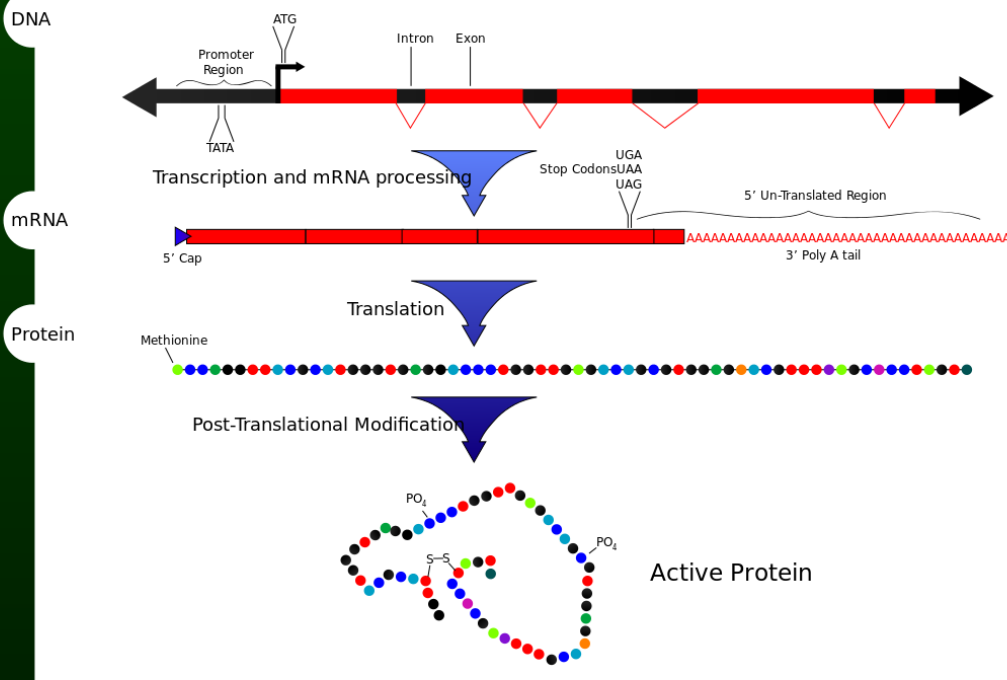
# DNA → AA

		Second Codon Letter				Third Codon Letter
		T	C	A	G	
First Codon Letter	T	TTT – phe TTC – phe TTA – leu TTG – leu	TCT – ser TCC – ser TCA – ser TCG – ser	TAT – tyr TAC – tyr TAA – stop TAG – stop	TGT – cys TGC – cys TGA – stop TGG – trp	
	C	CTT – leu CTC – leu CTA – leu CTG – leu	CCT – pro CCC – pro CCA – pro CCG – pro	CAT – his CAC – his CAA – gln CAG – gln	CGT – arg CGC – arg CGA – arg CGG – arg	
	A	ATT – ile ATC – ile ATA – ile ATG – start/met	ACU – thr ACC – thr ACA – thr ACG – thr	AAT – asn AAC – asn AAA – lys AAG – lys	AGT – ser AGC – ser AGA – arg AGG – arg	
	G	GTT – val GTC – val GTA – val GTG – val	GCT – ala GCC – ala GCA – ala GCG – ala	GAT – asp GAC – asp GAA – glu GAG – glu	GGT – gly GGC – gly GGA – gly GGG – gly	

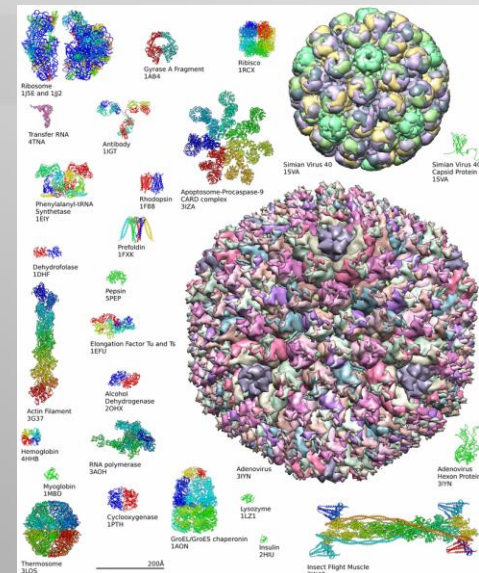


# Central Dogma

## Central Dogma of Molecular Biology : Eukaryotic Model



- DNA → RNA → Protein
- Transcription & Translation (& Replication)



# DNA & RNA

- Deoxyribonucleic acid

- Ribonucleic acid

- Base: A, T/U, C, G

- Adenine

- Thymine

- Uracil

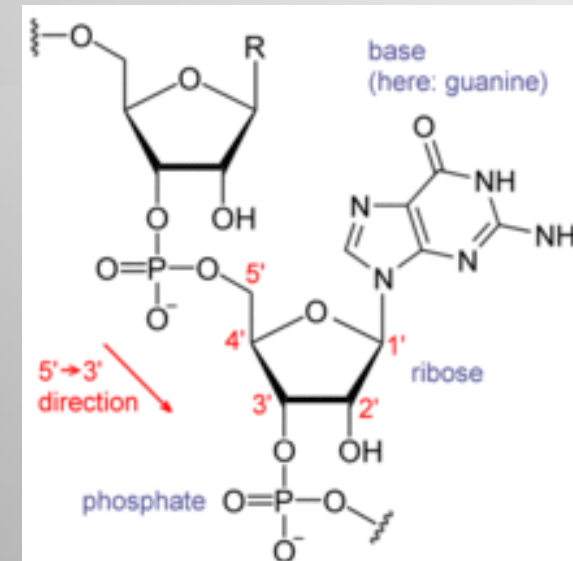
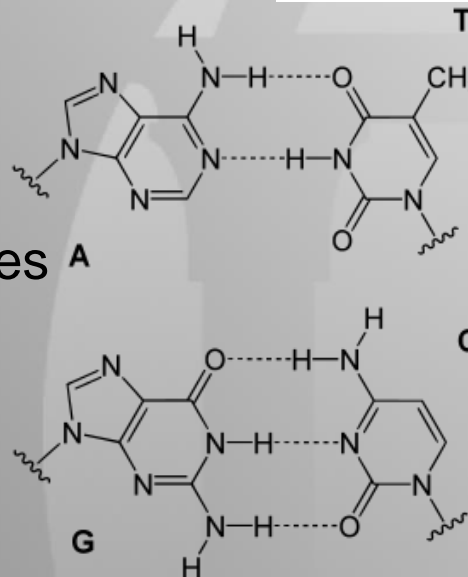
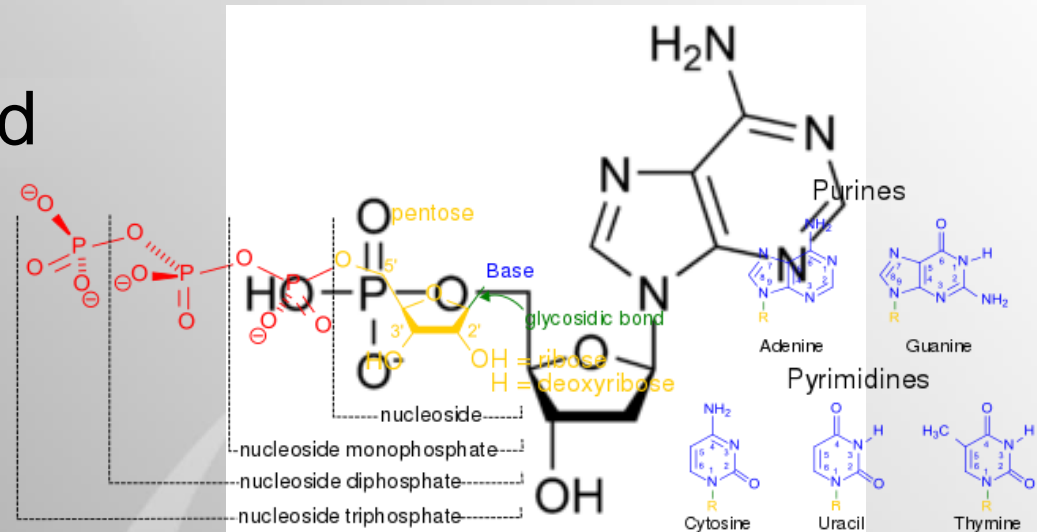
- Guanine

- Cytosine

- Purines & Pyrimidines

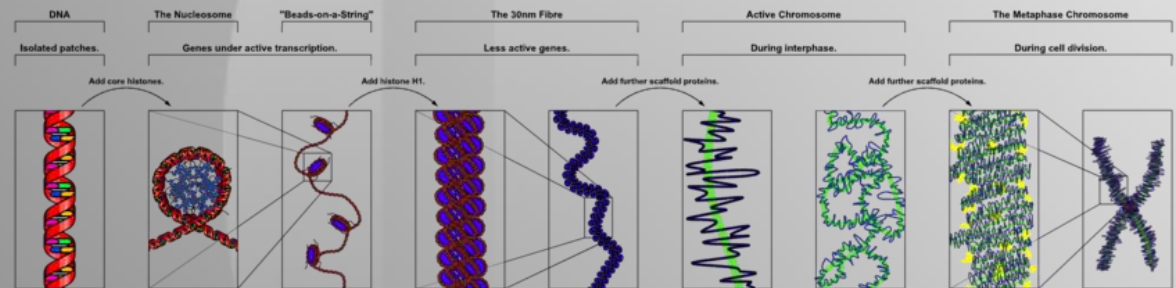
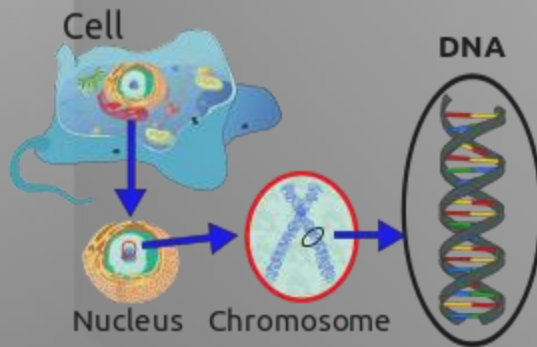
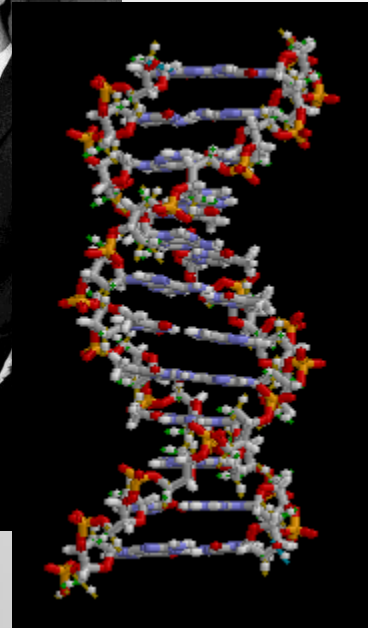
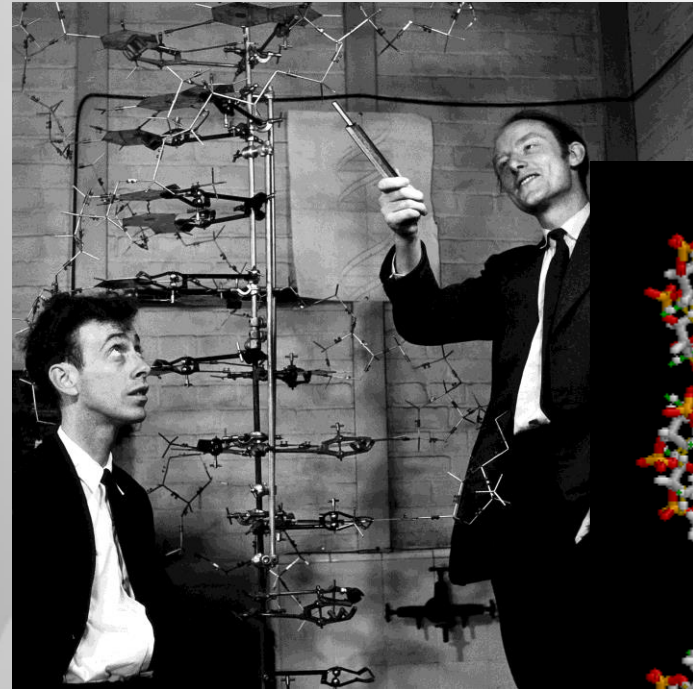
- Ribose

- Phosphate



# Double Helix!

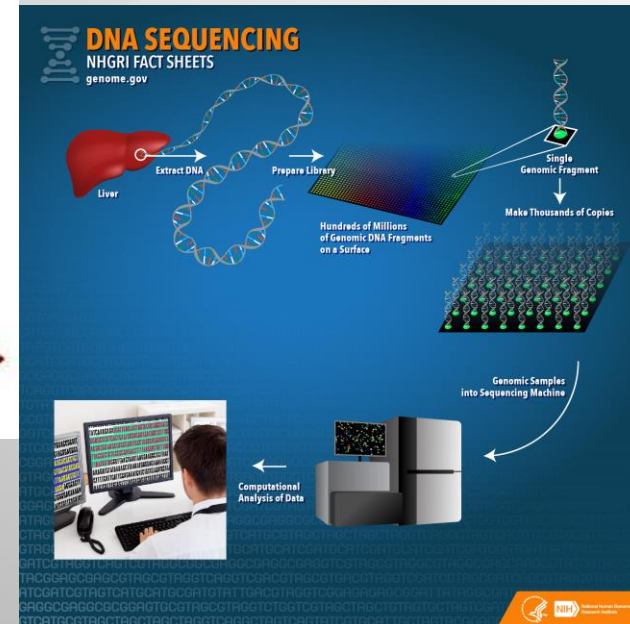
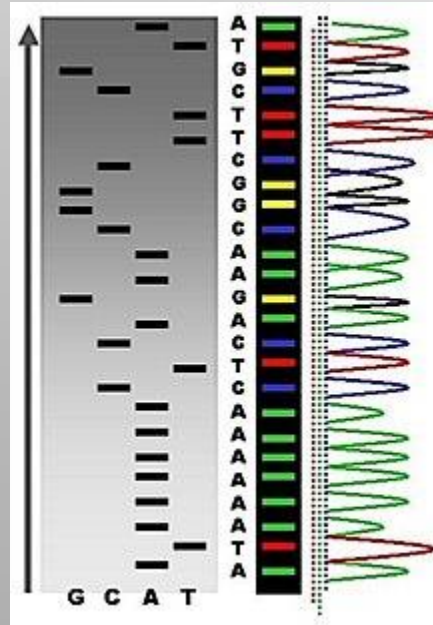
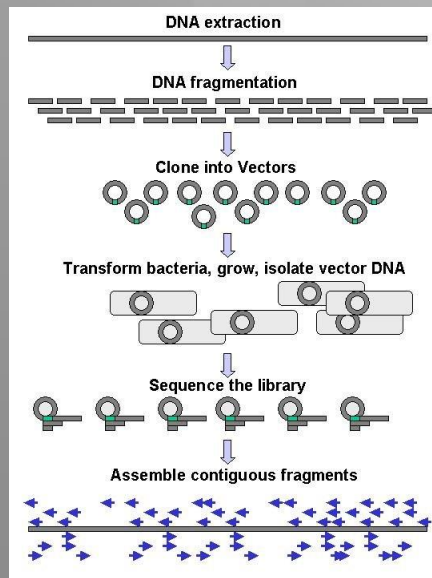
- Double helix
  - Chromosomes
- Replication
  - DNA-polymerase
- Transcription
- Translation





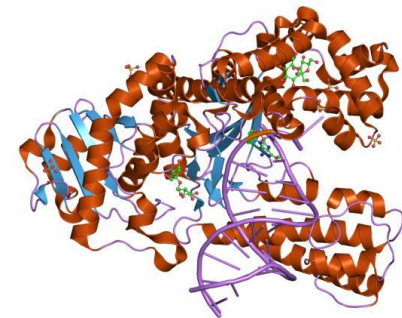
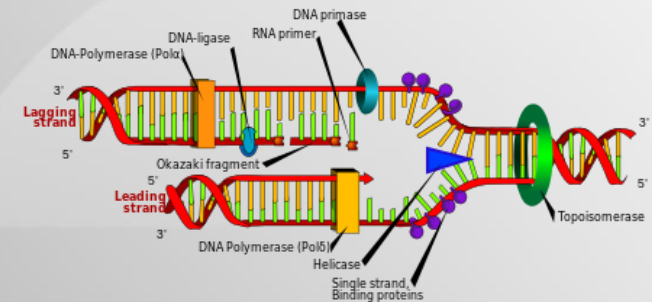
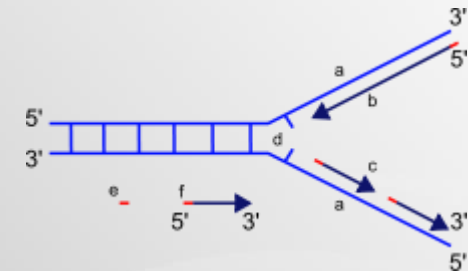
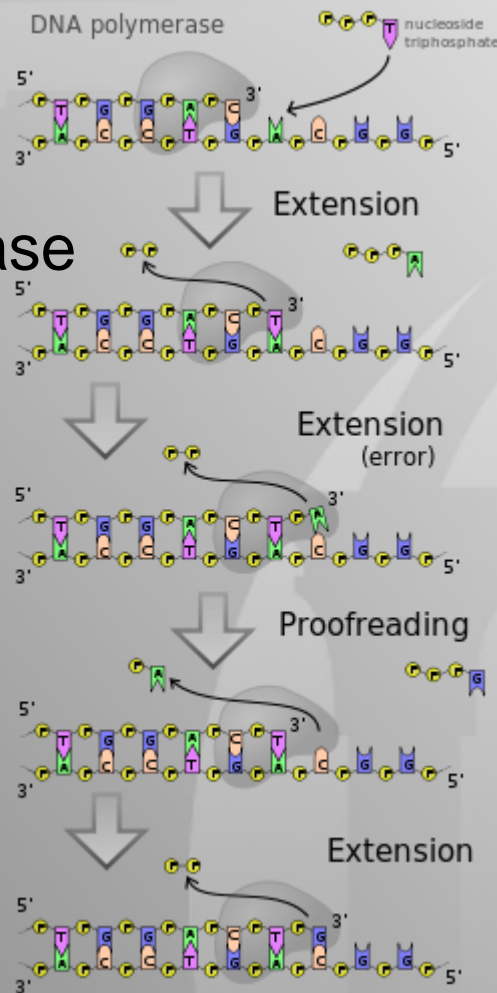
# Sequencing

- Restriction Enzyme
- Gel & Electrophoresis



# Replication

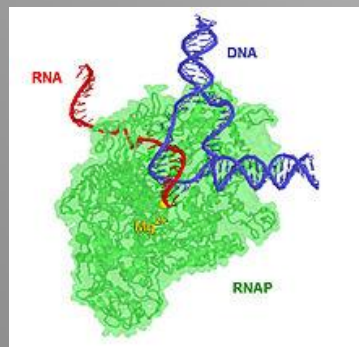
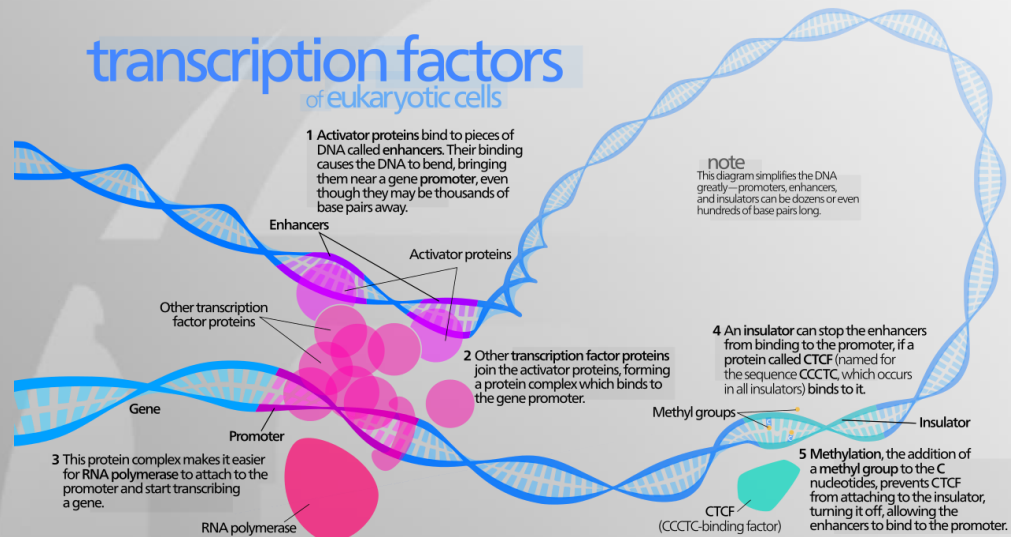
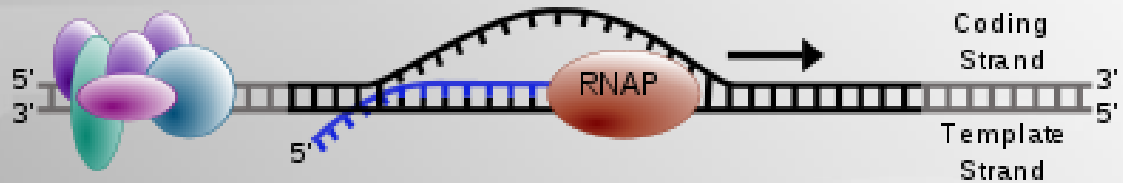
- Double helix
- Replication
  - DNA-polymerase
- Transcription
- Translation



# Transcription

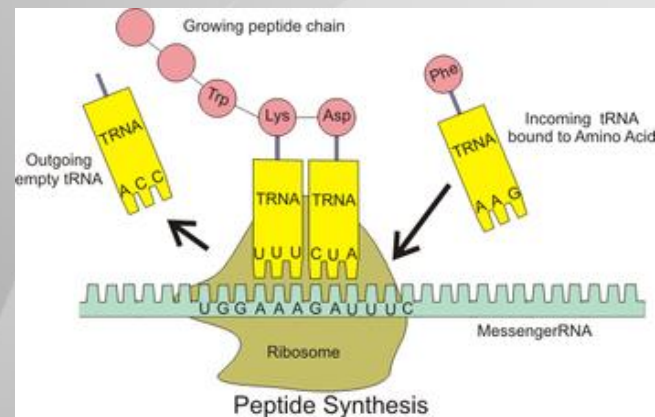
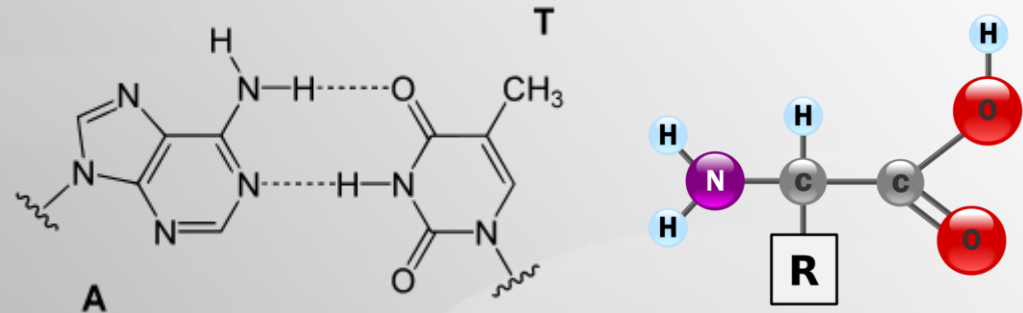


- Double helix
- Replication
- Transcription
  - RNA Polymerase
  - Transcription Factors
  - Expression control
- Translation



# Translation

- Double helix
- Replication
- Transcription
- Translation
  - Ribosome
  - mRNA
  - tRNA
  - Nucleic acid to Amino acid



		Second Codon Letter				
		T	C	A	G	
First Codon Letter	T	TTT - phe TTC - phe TTA - leu TTG - leu	TCT - ser TCC - ser TCA - ser TCG - ser	TAT - tyr TAC - tyr TAA - stop TAG - stop	TGT - cys TGC - cys TGA - stop TGG - trp	Third Codon Letter
	C	CTT - leu CTC - leu CTA - leu CTG - leu	CCT - pro CCC - pro CCA - pro CCG - pro	CAT - his CAC - his CAA - gln CAG - gln	CGT - arg CGC - arg CGA - arg CGG - arg	
	A	ATT - ile ATC - ile ATA - ile ATG - start/met	ACU - thr ACC - thr ACA - thr ACG - thr	AAT - asn AAC - asn AAA - lys AAG - lys	AGT - ser AGC - ser AGA - arg AGG - arg	
	G	GTT - val GTC - val GTA - val GTG - val	GCT - ala GCC - ala GCA - ala GCG - ala	GAT - asp GAC - asp GAA - glu GAG - glu	GGT - gly GGC - gly GGA - gly GGG - gly	

```
#dictionary
```

```
def main():
```

```
    f1 = open("M:\\tubulin_a.txt", "r")
```

```
    seq1 = f1.readline()
```

```
    f2 = open("M:\\tubulin_b.txt", "r")
```

```
    seq2 = f2.readline()
```

```
    aa1 = {}
```

```
    aa2 = {}
```

```
-----
```

```
-----  
for i in range(len(seq1)):  
    if seq1[i] in aa1:  
        aa1[seq1[i]]+=1  
    else:  
        aa1[seq1[i]]=1  
for i in range(len(seq2)):  
    if seq2[i] in aa2:  
        aa2[seq2[i]] += 1  
    else:  
        aa2[seq2[i]] = 1  
for i in aa1:  
    print(i,aa1[i],aa1[i]/len(seq1),aa2[i],aa2[i]/len(seq2))
```



# 2D Array

- `List = [[0]*n, for l in range(n)]`





