TIN SINH HỌC ĐẠI CƯƠNG
(Introduction to Bioinformatics)

PGS.TS. Trần Văn Lăng
Email: langtv@vast.vn

Chương 5:

NGÔN NGỮ PYTHON TRONG SINH TIN HỌC

NỘI DUNG

- Giới thiệu về Ngôn ngữ Python
- Sử dung Python cho một số thao tác thông dụng

GIỚI THIỆU VỀ NGÔN NGỮ PYTHON

- Python là ngôn ngữ lập trình hướng đối tượng; là một ngôn ngữ thông dịch.
- Source code của Python mã nguồn mở, do tổ chức phi lợi nhuận Python Software Foundation quản lý.

- Python được phát triển bởi Guido Van Rossum vào cuối những năm 80 và đầu những năm 90 tại Viện toán - tin ở Hà Lan.
- Python kế thừa từ nhiều ngôn ngữ như ABC, Module-3, C, C++, Unix Shell, …

- Ngôn ngữ Python được cập nhật khá thường xuyên để thêm các tính năng và hỗ trợ mới. Phiên bản mới nhất là **Python 3.6.4**.

- Download trực tiếp từ https://www.python.org/downloads/

SỬ DỤNG PYTHON TRONG ỨNG DỤNG SINH HỌC PHÂN TỬ

Ví dụ về trình tự DNA

- **Ví dụ 01**: Cho biết chiều dài của một trình tự DNA bất kỳ
- **Ví dụ 02**: Cho một trình tự DNA, cho biết có bao nhiêu Adenine, bao nhiêu Cytosine, Guanine, và Thymine

Ví dụ 01

- Sau khi khởi động Python, thực thi trực tiếp bằng các câu lệnh:
  - DNASeq = raw_input("Cho mot trinh tu DNA: ")
  - print "Trinh tu DNA nay co chieu dai la", len(DNASeq), "bases"

```
>>> DNASeq = raw_input( "Cho mot trinh tu DNA: " )
Cho mot trinh tu DNA: AAACGTACGTAATTGCCAATGAAGT
>>> print "Trinh tu DNA nay co chieu dai la ", len(DNASeq), " bases"
Trinh tu DNA nay co chieu dai la  25  bases
>>> _
```


Kiểu dữ liệu String

- **Để nhập dữ liệu** trong Python, có thể dùng hàm **raw_input()**, hoặc cũng có thể dùng hàm **input()**.
- Tuy nhiên, khi dùng **input()** phải nhập đúng quy định dạng chuỗi ký tự - phải có cặp dấu nháy đơn hoặc kép chứa chuỗi ký tự đó.

```
>>> DNA = input( "Cho mot trinh tu DNA: " )
Cho mot trinh tu DNA: AAACGTAA
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'AAACGTAA' is not defined
>>> DNA = input( "Cho mot trinh tu DNA: " )
Cho mot trinh tu DNA: "AAACGTAA"
>>>
```


- **Để biết chiều dài** của một trình tự, dùng hàm **len()**.
- Như bên cạnh, với trình tự DNASeq và DNA đã nhập vào, chiều dài là 25 và 8 tương ứng

```
>>> len( DNA )
8
>>> len( DNASeq )
25
```


Ví dụ 02

- **Để biết bao nhiêu loại** base trong một trình tự, dùng hàm **count()**

```
>>> print "Co", DNASeq.count("T"), "Thymine trong trinh tu", DNASeq
Co 6 Thymine trong trinh tu AAACGTACGTAATTGCCAATGAAGT
>>>
```



```
>>> print "Co", DNASeq.count("A"), "Adenine,", DNASeq.count("C"), "Cytosine va", DNASeq.count("G"), "Guanine"
Co 10 Adenine, 4 Cytosine va 5 Guanine
>>> _
```

- Ở đây dùng hàm **count()** để đếm.
- Hàm này sử dụng bằng cách chỉ tên của đối tượng ở phía trước, còn ký tự cần đếm được viết trong danh sách đối số của hàm

- Trường hợp trình tự nhập vào vừa có chữ hoa, vừa có chữ thường, phải chuyển đổi để trở thành thống nhất trước khi đếm bằng hàm **upper()** hoặc **lower() để chuyển thành chữ hoa** hoặc **chữ thường**.


```
>>> DNA2 = "GGTCCAAaaatgCTTGacttTTTgCAAtg"
>>> DNA2.upper()
'GGTCCAAAAATGCTTGACTTTTTGCAATG'
>>> DNA2.lower()
'ggtccaaaaatgcttgactttttgcaatg'
>>> _
```

- Khi đó có thể đếm số base Adenine trong trình tự DNA2 bằng: DNA2.upper().count("A")

Ví dụ 03

- **Ví dụ 03**: Tính tỷ lệ phần trăm của GC Content trong một trình tự DNA

```
>>>
>>> c = DNASeq.count("C")
>>> g = DNASeq.count("G")
>>> s = len(DNASeq)
>>> print "GC Content = ", (c+g+0.0)/s*100, "%"
GC Content =  36.0 %
>>>
```


Về chuỗi ký tự

- Với chuỗi ký tự S; phần tử đầu tiên là S[0] hoặc S[-len(S)], phần tử cuối cùng là S[len(S)-1] hoặc S[-1]
- Thí dụ: vị trí chuỗi S như sau:

T	V	L	A	N	G
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

- Một chuỗi con trong S được truy cập bằng S[x:y:stride],
  - trong đó x là vị trí bắt đầu, y là vị trí gần kề và stride là bước dài (default là 1).
  - Thí dụ: chuỗi con gồm 4 ký tự, bắt đầu từ ký tự thứ III của S:

    S[2:6:1] hoặc S[2:6] là "LANG"

  - Chuỗi con từ ký tự thứ II cho đến hết chuỗi:

S[1:], S[1::], S[1::1], S[1:len(S)], S[1:len(S):1] là "VLANG"

  - Với T = "123456789", T[-7:-3] là "3456" (cũng là T[2:6], T[-7:-3:1], T[2:6:1])
- Khi stride có giá trị âm, thứ tự theo chiều ngược lại.
  - Thí dụ: T[::-1] là "987654321", T[-4:-8:-1] là "6543" (cũng là T[5:1:-1]

- Lưu ý:
  - stride là số dương, thì phải có x < y (nếu không sẽ là chuỗi rỗng)
  - stride là số âm, thì phải có x > y (nếu không sẽ là chuỗi rỗng)

Ví dụ 04

- **Ví dụ 04**: Tìm trình tự đảo ngược (Reverse Sequence) của trình tự DNA cho trước.

```
>>> DNASeq="ACGTCTGATG"
>>> Reverse=DNASeq[::-1]
>>> Reverse
'GTAGTCTGCA'
>>> _
```

- **Ví dụ 05**: Tìm trình tự bổ sung đảo ngược (Reverse Complement Sequence) của một trình tự DNA trong cặp trình tự DNA

- DNA là một cặp trình tự, trong đó trình tự thứ nhất (ký hiệu 3' – 5') và trình tự thứ hai (ký hiệu 5' – 3') gọi là trình tự bổ sung (Complement Sequence). Chúng liên kết với nhau theo nguyên tắc A với T, C với G.

- Lần lượt thay A bởi ký hiệu trung gian x, sau đó thay T bởi A, rồi tiếp tục chuyển x thành T
- Tương tự như vậy với G và C có được trình tự bổ sung

```
>>> DNA53="ACGTACGT"
>>> Temp=DNA53.replace('A','x').replace('T','A').replace('x','T')
>>> DNA35=Temp.replace('C','x').replace('G','C').replace('x','G')
>>> DNA35
'TGCATGCA'
>>>
```


- Sau đó, tìm trình tự bổ sung đảo ngược (Reverse Complement Sequence) bằng cách đảo ngược trình tự bổ sung

```
>>> DNA53="ACGTACGTCG"
>>> Temp=DNA53.replace('A','x').replace('T','A').replace('x','T')
>>> DNA35=Temp.replace('C','x').replace('G','C').replace('x','G')
>>> DNA35
'TGCATGCAGC'
>>> DNA35[::-1]
'CGACGTACGT'
>>> _
```

- **Ví dụ 06**: Phiên mã trình tự DNA sang trình tự RNA

Ví dụ 06

- Trình tự RNA có được bằng cách thay Thymine trong trình tự DNA bằng Uracile.
- Trong Python dùng hàm replace()

```
>>>
>>> print "Trinh tu RNA cua trinh tu", DNASeq, "la", DNASeq.replace("T","U")
Trinh tu RNA cua trinh tu AAACGTACGTAATTGCCAATGAAGT la AAACGUACGUAAUUGCCAAUGAAGU
>>>
```


- Cũng có thể lấy trình tự bổ sung (5' – 3') để thay Thymine bằng Uracile.

```
>>> DNA53="ACGTACGTCG"
>>> Temp=DNA53.replace('A','x').replace('T','A').replace('x','T')
>>> DNA35=Temp.replace('C','x').replace('G','C').replace('x','G')
>>> RNA=DNA35[::-1].replace('T','U')
>>> RNA
'CGACGUACGU'
>>>
```


- **Ví dụ 07**: Tìm vị trí của start codon và stop condon trong một trình tự RNA.
  - Biết rằng start condon là AUG,
  - Stop codon là UAA hoặc UAG hoặc UGA

Ví dụ 07

```
>>>
>>> print "Start condon cua", DNASeq, "o vi tri", DNASeq.replace("T","U").find("AUG")
Start condon cua AAACGTACGTAATTGCCAATGAAGT o vi tri 18
>>>
```

- Trên cơ sở Ví dụ 07, có thể mở rộng để có **Ví dụ 08**: Chỉ ra trình tự RNA trong đoạn từ start đến stop condon:

Ví dụ 08

```
>>> Gen =
"TGCTTATCCGGGCAGAACTAAGCCAGCCTGGCGCTCTCCTAGGGGACGACCAGATT
TATAATGTAATCGTTACAGCACACGCTTTCGTAATAATTTTCTTTATAGTAATGCCAATT
ATGATTGGAGGGTTTGGAAACTGACTAATCCCCCTAATGATCGGCGCCCCCGATATG
GCCTTCCCTCGAATAAATAACATAAGCTTTTGACTCCTACCTCCTTCGTTCCTTCTTCT
CTTAGCGTCTTCTGGCGTAGAAGCAGGGGCCGGAACTGGATGAACCGTCTATCCTC
CTCTAGCCAGCAACCTAGCACATGCCGGAGCATCAGTTGACCTTACAATTTTCTCCCT
TCACCTGGCAGGTGTCTCCTCAATTTTAGGTGCTATTAACTTCATTACTACTATTATTA
ACATGAAACCTCCCGCAATTTCCCAGTACCAAACCCCACTCTTCGTATGGGCTGTTCT
TATTACTGCCGTTCTCCTGCTTCTATCCCTGCCAGTTCTCGCTGCCGGAATTACCATG
CTTTTAACAGATCGAAACTTAAACACTTCTTTCTTCGACCCAGCAGGAGGAGGGGATC
CTATTCTATACCAGCACCT"
>>> mRNA = Gen.replace("T","U")
>>> mRNA[mRNA.find("AUG"):mRNA.find("UGA")]
'AUGUAAUCGUUACAGCACACGCUUUCGUAAUAAUUUUCUUUAUAGUAAUGCCAAU
UA'
```


Lệnh lặp for, kiểu list và dictionary

- Lệnh lặp **for** dùng với phép toán **in**.
- Ví dụ:

```
>>> RNA
'UGCAUGCAGC'
>>> for base in RNA:
...     print "Base: ", base
...
Base:  U
Base:  G
Base:  C
Base:  A
Base:  U
Base:  G
Base:  C
Base:  A
Base:  G
Base:  C
```


- Kiểu dữ liệu **list** được khai báo bằng cặp dấu **[]**.
- Ví dụ: chuyển một trình tự RNA thành một list

```
>>> RNA
'UGCAUGCAGC'
>>> list = []
>>> for base in RNA:
...     list.append(base)
...
>>> list
['U', 'G', 'C', 'A', 'U', 'G', 'C', 'A', 'G', 'C']
>>> _
```

- Có thể chuyển đổi một list thành một string (chuỗi) bằng hàm **join()**. Chẳng hạn, để thêm ký tự gap (dấu '-') vào

```
>>> list
['U', 'G', 'C', 'A', 'U', 'G', 'C', 'A', 'G', 'C']
>>> "-".join(list)
'U-G-C-A-U-G-C-A-G-C'
>>> _
```


- Kiểu **Dictionary** được tạo trong cặp **{}** với dạng **{keys, values}**.
- Ví dụ: Tìm trình tự bổ sung của trình tự 5'-3'

```
>>> dict = {"A": "T", "T": "A", "C": "G", "G": "C"}
>>> DNA53="ACGTACGTCG"
>>> list=[]
>>> for base in DNA53:
...     list.append( dict[base] )
...
>>> DNA35="".join(list)
>>> DNA35
'TGCATGCAGC'
>>>
```


Hàm **range()**

- Cấu trúc

  range([start,] stop [, step])
- Trong đó,
  - start: giá trị bắt đầu, default là 0
  - stop: giá trị kết thúc (nhỏ hơn giá trị này)
  - step: bước nhảy, default là 1

- Ví dụ: Tính bình phương của các số từ [0,4]

```
>>> for i in range(5):
...     print i, "bình phương là", i*i
...
0 bình phương là 0
1 bình phương là 1
2 bình phương là 4
3 bình phương là 9
4 bình phương là 16
>>> _
```

- Ví dụ: Tính bình phương của các số 2, 4, 6, 8

```
>>> for i in range(2,10,2):
...     print "Bình phương của", i, "là", i*i
...
Bình phương của 2 là 4
Bình phương của 4 là 16
Bình phương của 6 là 36
Bình phương của 8 là 64
>>> _
```


Tổ chức chương trình

- Trong Python, có thể thực thi các câu lệnh một cách trực tiếp như trên.
- Tuy nhiên, khi có nhiều câu lệnh cần soạn thảo trước; chúng ta có thể lưu trữ vào trong một tập tin Python (có phần mở rộng là .py) để gọi thực hiện một lần bang câu lệnh sau tại dấu nhắc của hệ điều hành:

python example.py

- Ví dụ: chương trình tìm trình tự bổ sung đảo ngược có tên DNA35.py thực hiện như sau:

```
Lion:python lang$ python DNA35.py
Cho mot trinh tu DNA:  ACGTGGGACCCCTC
DNA bo sung dao nguoc: GAGGGGTCCCACGT
Lion:python lang$ _
```


- Trong đó, tập tin DNA35.py có nội dung là:

```
# Reverse Complement Sequence
# Author: A.Prof. Tran Van Lang, PhD.
# Name:   DNA35.py

DNA53 = raw_input( "Cho mot trinh tu DNA:  " )
DNA35 = DNA53.replace('A','y').replace('T','A').replace('y','T')
DNA35 = DNA35.replace('C','y').replace('G','C').replace('y','G')
print "DNA bo sung dao nguoc:", DNA35[::-1]

# The End
```

- Để tập tin DNA35.py có thể thi hành được mà không cần gọi đến Python, phải thêm dòng sau đây vào đầu tập tin:

  #! /usr/bin/python

- Trong đó /usr/bin là folder chứa python
- Sau đó, them thuộc tính thi hành (execute) cho tập tin này bang dòng lệnh

  chmod +x DNA35.py

- Nội dung tập tin mới là:

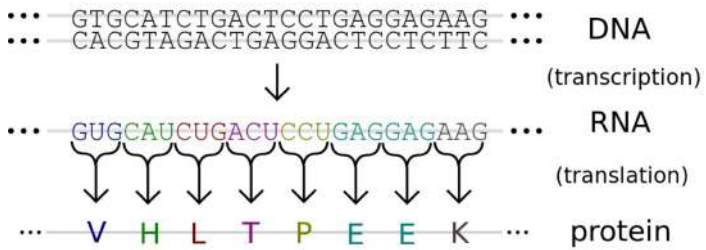
```
#! /usr/bin/python

# Reverse Complement Sequence
# Author: A.Prof. Tran Van Lang, PhD.
# Name:   DNA35.py

DNA53 = raw_input( "Cho mot trinh tu DNA:  " )
DNA35 = DNA53.replace('A','y').replace('T','A').replace('y','T')
DNA35 = DNA35.replace('C','y').replace('G','C').replace('y','G')
print "DNA bo sung dao nguoc:", DNA35[::-1]

# The End
```


- Các câu lệnh thi hành, sau khi đã change mode bang lệnh

  chmod +x DNA35.py

```
Lion:python lang$ ./DNA35.py
Cho mot trinh tu DNA:  AACGTTACGT
DNA bo sung dao nguoc: ACGTAACGTT
Lion:python lang$
```


- **Ví dụ 09**: Dịch mã trình tự RNA sang trình tự protein.



Ví dụ 09

- Để giải quyết vấn đề này cần 2 bước chính:
  - Bước 1: tách thành các codon riêng
  - Bước 2: "tra tự điển" để dịch mã

- **Bước 1**: Với một trình tự RNA có được, tách thành các codon tương ứng

```
>>> RNA="UGCAUCGGACCGAUUGAU"
>>> for b in range(0,len(RNA),3):
...     print "Codon", b/3+1, "là", RNA[b:b+3]
...
Codon 1 là UGC
Codon 2 là AUC
Codon 3 là GGA
Codon 4 là CCG
Codon 5 là AUU
Codon 6 là GAU
>>>
```


- **Bước 2**: Tạo tự điển các amino acid

```
dict_aa1 = {            # leter '#' is stop codon
        "UUU":'F',"UUC":'F',"UUA":'L',"UUG":'L',
        "CUU":'L',"CUC":'L',"CUA":'L',"CUG":'L',
        "AUU":'I',"AUC":'I',"AUA":'I',"AUG":'M',
        "GUU":'V',"GUC":'V',"GUA":'V',"GUG":'V',
        "UCU":'S',"UCC":'S',"UCA":'S',"UCG":'S',
        "CCU":'P',"CCC":'P',"CCA":'P',"CCG":'P',
        "ACU":'T',"ACC":'T',"ACA":'T',"ACG":'T',
        "GCU":'A',"GCC":'A',"GCA":'A',"GCG":'A',
        "UAU":'Y',"UAC":'Y',"UAA":'#',"UAG":'#',
        "CAU":'H',"CAC":'H',"CAA":'G',"CAG":'G',
        "AAU":'N',"AAC":'N',"AAA":'K',"AAG":'K',
        "GAU":'D',"GAC":'D',"GAA":'E',"GAG":'E',
        "UGU":'C',"UGC":'C',"UGA":'#',"UGG":'W',
        "CGU":'R',"CGC":'R',"CGA":'R',"CGG":'R',
        "AGU":'S',"AGC":'S',"AGA":'R',"AGG":'R',
        "GGU":'G',"GGC":'G',"GGA":'G',"GGG":'G',
}
```



```
dict_aa3 = {
        "UUU":'Phe',"UUC":'Phe',"UUA":'Leu',"UUG":'Leu',
        "CUU":'Leu',"CUC":'Leu',"CUA":'Leu',"CUG":'Leu',
        "AUU":'Ile',"AUC":'Ile',"AUA":'Ile',"AUG":'Met',
        "GUU":'Val',"GUC":'Val',"GUA":'Val',"GUG":'Val',
        "UCU":'Ser',"UCC":'Ser',"UCA":'Ser',"UCG":'Ser',
        "CCU":'Pro',"CCC":'Pro',"CCA":'Pro',"CCG":'Pro',
        "ACU":'Thr',"ACC":'Thr',"ACA":'Thr',"ACG":'Thr',
        "GCU":'Ala',"GCC":'Ala',"GCA":'Ala',"GCG":'Ala',
        "UAU":'Tyr',"UAC":'Tyr',"UAA":'Stop',"UAG":'Stop',
        "CAU":'His',"CAC":'His',"CAA":'Gln',"CAG":'Gln',
        "AAU":'Asn',"AAC":'Asn',"AAA":'Lys',"AAG":'Lys',
        "GAU":'Asp',"GAC":'Asp',"GAA":'Glu',"GAG":'Glu',
        "UGU":'Cys',"UGC":'Cys',"UGA":'Stop',"UGG":'Trp',
        "CGU":'Arg',"CGC":'Arg',"CGA":'Arg',"CGG":'Arg',
        "AGU":'Ser',"AGC":'Ser',"AGA":'Arg',"AGG":'Arg',
        "GGU":'Gly',"GGC":'Gly',"GGA":'Gly',"GGG":'Gly',
}
```

- Chương trình có dạng

```
#! /usr/bin/python
# Translation RNA to Protein
# Author: A.Prof. Tran Van Lang, PhD.
# Name:   RNA2Protein.py

dict_aa1 = {            # leter '#' is stop codon
        #######
}
dict_aa3 = {
        #######
}
RNA = "UCGGGCUACGAACGUACGGCCCUUUGC"
protein1 = ""
protein3 = ""
for b in range(0,len(RNA),3):
        protein1 += dict_aa1[RNA[b:b+3]]
        protein3 += dict_aa3[RNA[b:b+3]]
print "With RNA sequence:", RNA
print "Protein in 1-letter code:", protein1
print "Protein in 3-letter code:", protein3
# The End
```


Tổ chức chương trình

- Cấu trúc của một chương trình Python cũng có thể bao gồm nhiều:
  - chương trình con
  - file liên kết với nhau

Chương trình con

- Chương trình con (Function) là một khối các câu lệnh được gom lại với một tên gọi thông qua từ khóa **def**.
- Chẳng hạn, chương trình con xuất ra trình tự RNA từ trình tự DNA cho trước.

```
def ToRNA( DNA ):
        RNA = DNA.replace('T','U')
        return RNA
```


- Khi gọi chương trình con, sử dụng tên của chương trình con kèm theo danh sách tham số

```
>>> def ToRNA( DNA ):
...     RNA = DNA.replace('T','U')
...     return RNA
...
>>> DNA = "AACGTCCGTAG"
>>> RNA = ToRNA(DNA)
>>> RNA
'AACGUCCGUAG'
```

Một chương trình gồm nhiều file

- Các file liên kết với nhau qua từ khóa **import** cùng với tên file tương ứng.
- Ví dụ: Chương trình con **ToRNA()** được viết trong file có tên **Function.py**

```
# Our Python Libraries
# By A.Prof Tran Van Lang, PhD
# Filename: Function.py

##### Transcription DNA to RNA
def ToRNA( DNA ):
        RNA = DNA.replace('T','U')
        return RNA

# The End
```


- Để gọi sử dung từ một file khác (chẳng hạn **Using.py**). Phải **import** như là sự liên kết, và gọi hàm phải chỉ định thêm tên file chứa hàm:

```
# Using Libraries from other files
# By A.Prof Tran Van Lang, PhD
# Filename: Using.py

import Function

print Function.ToRNA( "AAAACGTTCGTAGCT" )
```


- Trong việc tìm trình tự bổ sung, có thể viết:

```
# Our Python Libraries
# By A.Prof Tran Van Lang, PhD
# Filename: Function.py

def ReverseComplement2( DNA ):
        for i in range(len(DNA)):
                if DNA[i] == 'A':
                        DNA = DNA[:i] + 'T' + DNA[i+1:]
                elif DNA[i] == 'T':
                        DNA = DNA[:i] + 'A' + DNA[i+1:]
                elif DNA[i] == 'G':
                        DNA = DNA[:i] + 'C' + DNA[i+1:]
                elif DNA[i] == 'C':
                        DNA = DNA[:i] + 'G' + DNA[i+1:]
        return DNA[::-1]
```


- Sau đó gọi hàm bằng cách

```
# Finding the Reverse Complement Sequence
# By A.Prof Tran Van Lang, PhD
# Filename: bio.py

import Function as fc

DNA = raw_input( "Enter a DNA Sequence: " )
print "Reverse Complement: ", fc.ReverseComplement2( DNA )
```

Kiểu File trong Python

- Khi dữ liệu nhiều, có thể tổ chức để lưu trữ trong một gập tin.
- Sau đó lấy dữ liệu ra bằng hàm **readline()**, **readlines()** hoặc để ghi lên bằng hàm **writeline()**, **writelines()**

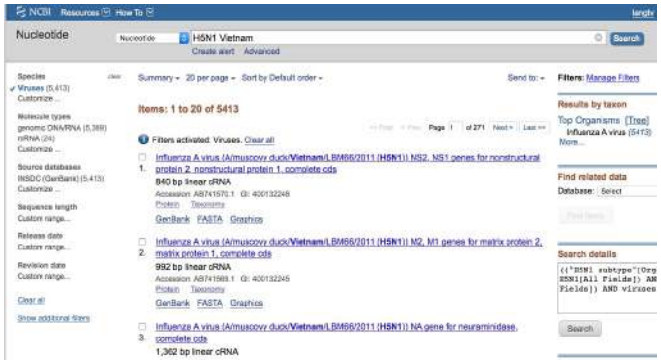
- **Ví dụ 10**: Sao chép nội dung tập tin Using.py thành tập tin Using.save

```
open("Using.save",'w').writelines(open("Using.py",'r')
```

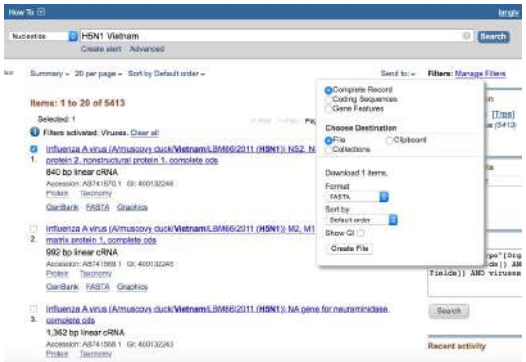

- **Ví dụ 11**: Lấy dữ liệu về Nucleotide từ NCBI dưới dạng file FASTA. Sau đó lắp ghép các dòng Nucleotide có trong file này để tạo ra trình tự DNA tương ứng

- Cần tìm trên NCBI về Nucleotide của H5N1 Vietnam.

- Sau khi chọn "Search" có kết quả



- Chọn Send to để lấy kết quả thứ I, và Create File FASTA



- Khi đó, trên đĩa có tập tin sequence.fasta.txt với nội dung. Trong đó
  - Dòng thứ I: mô tả tên của dữ lieu
  - Các dòng còn lại: trình tự DNA thành các dòng

sequence.fasta.txt

```
>AB741570.1 Influenza A virus (A/muscovy duck/Vietnam/LBM66/2011 (H5N1)) NS2, NS1 genes
for nonstructural protein 2, nonstructural protein 1, complete cds
ACAAAAACATAATGGATTCCAACACTGTTTCAAGCTTTCAGGTAGACTGCTTTCTTTGGCATGTCCGCAA
ACGATTTGCAGACCAAGAACTGGGTGATGCCCCATTCCTTGACCGGCTTCGCCGAGATCAGAAGTCCCTA
AGAGGAAGAGGCAACACTCTTGGTCTGGACATCGAAACAGCTACTCGTGCGGGAAAACAGATAGTGGAGC
GGATTCTTGATGAGGAACCTGATGAGGCACTTAAAATGCCGACTTCACGCTACCTAACTGAAATGACTCT
CGAAGAAATGTCAAGGGACTGGTTCATGCTCATGCCCAAGCAGAAAGTGGTGGGTTCCCTTTGCATCAAA
ATGGACCAGGCAATAATGGATAAAACCATCATACTGAAAGCAAACTTCAGTGTGATTTTTGACCGGTTAG
AGACCCTAATACTACTCAGAGCTTTCACAGAAGAAGGAGCAATCGTGGGGGAAATCTCACCATTACCTTC
TCTTCCAGGACATACTGGTGAGGATGTCAAAAATGCAATTGGCGTCCTCATCGGAGGACTTGAATGGAAT
GATAACACAGTTCGGGTCTCTGAAACTGTACAGAGATTCGCTTGGAGAAGCAGTGATGAGAGTGGGAGAC
TTCCACTCCCTCCAAATCAGAAACGGAAATTGGCGAGAGCAATTGGGTCAGAAGTTTGAAGAAATAAGGT
GGCTGATTGAAGAAGTACGACATAGATTGAAAATTACAGAAAACAGCTTCGAACAGATAACGTTTATGCA
AGCCTTACAACTACTGCTTGAAGTGGAGCAAGAGATAAGAGCCTTCTCGTTTCAGCTTATTTAATGATAA
```


- Tạo một dòng là trình tự DNA từ file dạng fasta này

```
# Create DNA sequence from FASTA file
# By A.Prof Tran Van Lang, PhD
# Filename: FromFASTA.by

fp = open( "sequence.fasta.txt",'r' )
line1 = fp.readline()
dataname = line1[1:-1] # From pos #1 to pos #-1
DNA = ""
while True:
        line = fp.readline()
        if line == "":
                break
        DNA += line.replace( "\n", "" )

print "Du lieu:", dataname
print "Trinh tu DNA:", DNA
fp.close()

# The End
```

- Kết quả trình tự DNA được tạo ra

```
Lion:python lang$ python FromFASTA.py
Du lieu: AB741570.1 Influenza A virus (A/muscovy duck/Vietnam/LBM66/2011 (H5N1)) NS2, NS1 genes for nonstructural
protein 2, nonstructural protein 1, complete cds
Trinh tu DNA: ACAAAAACATAATGGATTCCAACACTGTTTCAAGCTTTCAGGTAGACTGCTTTCTTTGGCATGTCCGCAAACGATTTGCAGACCAAGAACTGGGTGATGC
CCCATTCCTTGACCGGCTTCGCCGAGATCAGAAGTCCCTAAGAGGAAGAGGCAACACTCTTGGTCTGGACATCGAAACAGCTACTCGTGCGGGAAAACAGATAGTGGAGCGGAT
TCTTGATGAGGAACCTGATGAGGCACTTAAAATGCCGACTTCACGCTACCTAACTGAAATGACTCTCGAAGAAATGTCAAGGGACTGGTTCATGCTCATGCCCAAGCAGAAAGT
GGTGGGTTCCCTTTGCATCAAAATGGACCAGGCAATAATGGATAAAACCATCATACTGAAAGCAAACTTCAGTGTGATTTTTGACCGGTTAGAGACCCTAATACTACTCAGAGC
TTTCACAGAAGAAGGAGCAATCGTGGGGGAAATCTCACCATTACCTTCTCTTCCAGGACATACTGGTGAGGATGTCAAAAATGCAATTGGCGTCCTCATCGGAGGACTTGAATG
GAATGATAACACAGTTCGGGTCTCTGAAACTGTACAGAGATTCGCTTGGAGAAGCAGTGATGAGAGTGGGAGACTTCCACTCCCTCCAAATCAGAAACGGAAATTGGCGAGAGC
AATTGGGTCAGAAGTTTGAAGAAATAAGGTGGCTGATTGAAGAAGTACGACATAGATTGAAAATTACAGAAAACAGCTTCGAACAGATAACGTTTATGCAAGCCTTACAACTAC
TGCTTGAAGTGGAGCAAGAGATAAGAGCCTTCTCGTTTCAGCTTATTTAATGATAA
Lion:python lang$ _
```


Hoặc cần lưu lại trình tự này trong tập tin có tên lấy từ dataname

```
# Create DNA sequence from FASTA file
# By A.Prof Tran Van Lang, PhD
# Filename: FromFASTA.by

fp = open( "sequence.fasta.txt",'r' )
line1 = fp.readline()
dataname = line1[1:-1] # From pos #1 to pos #-1
DNA = ""
while True:
        line = fp.readline()
        if line == "":
                break
        DNA += line.replace( "\n", "" )

print "Du lieu:", dataname
print "Trinh tu DNA:", DNA
fp.close()
open( dataname[:10]+".fasta",'w' ).writelines( DNA )
# The End
```


- **Ví dụ 12**: Bắt cặp 2 trình tự bằng thuật toán Needleman – Wunsch.

Ví dụ 12

- Có ba hàm (chương trình con)
  - Sigma: tính giá trị σ_{ij}
  - Alignment: tìm vết và bắt cặp
  - Score: tính điểm

- Hàm sigma()

```
def sigma( u,v,ma,mi,d ):
        if u == v:
                return ma
        elif u == '-' or v == '-':
                return d
        else:
                return mi
```


- Hàm alignment()

```
def alignment(U,V,ma,mi,d):
        n, m = len(V), len(U)
        AU, AV = "","
        M = numpy.zeros((n+1,m+1),dtype = int)
        for i in range(n+1):
                M[i][0] = i * d
        for j in range(m+1):
                M[0][j] = j * d

        for i in range(1,n+1):
                for j in range(1,m+1):
                        first  = M[i-1][j-1] + sigma(U[j-1],V[i-1],ma,mi,d)
                        second = M[i-1][j] + d
                        third  = M[i][j-1] + d
                        M[i][j] = max( first,second,third )
```



```
i, j = n, m
while i > 0 and j > 0:
        if M[i][j] == M[i-1][j-1] + sigma(U[j-1],V[i-1],ma,mi,d):
                AU += U[j-1]
                AV += V[i-1]
                i -= 1
                j -= 1
        elif M[i][j] == M[i-1][j] + d:
                AV += V[i-1]
                AU += "-"
                i -= 1
        elif M[i][j] == M[i][j-1] + d:
                AV += "-"
                AU += U[j-1]
                j -= 1
while i > 0 :
        AV += V[i-1]
        AU += "-"
        i -= 1
while j > 0:
        AV += "-"
        AU += U[j-1]
        j -= 1

print U
print V
print M
print AU[::-1]
print AV[::-1]
score( AU,AV,2,-1,-2 )
```


- Hàm score()

```
def score(AU,AV,ma,mi,d):
        nma = 0
        nmi = 0
        nd = 0
        n = len(AU)
        for i in range(n):
                if AU[i] == AV[i]:
                        nma += 1
                elif AU[i] == '-' or AV[i] == '-':
                        nd += 1
                nmi = n - (nma + nd)
        s = nma*ma + nmi*mi + nd*d
        print "Diem danh gia: %d(%d) + %d(%d) + %d(%d) = %d" % (nma,ma,nmi,mi,nd,d,s)
        return s
```

File NeedlmanWunschGap.py

- **Ví dụ 13**: Bắt cặp cục bộ bởi thuật toán Smith–Waterman.