Bioinformatics

Introduction

Section 1

Introduction to this course

List of Topics

- Introduction
- Introduction to programming in bioinformatics
- Basics of molecular biology
- Biological data banks
- Sequence analysis

What is Bioinformatics?

Bioinformatics is the use of computers for the acquisition, management, and analysis of biological information

It incorporates elements of molecular biology, computational biology, database computing, and the Internet

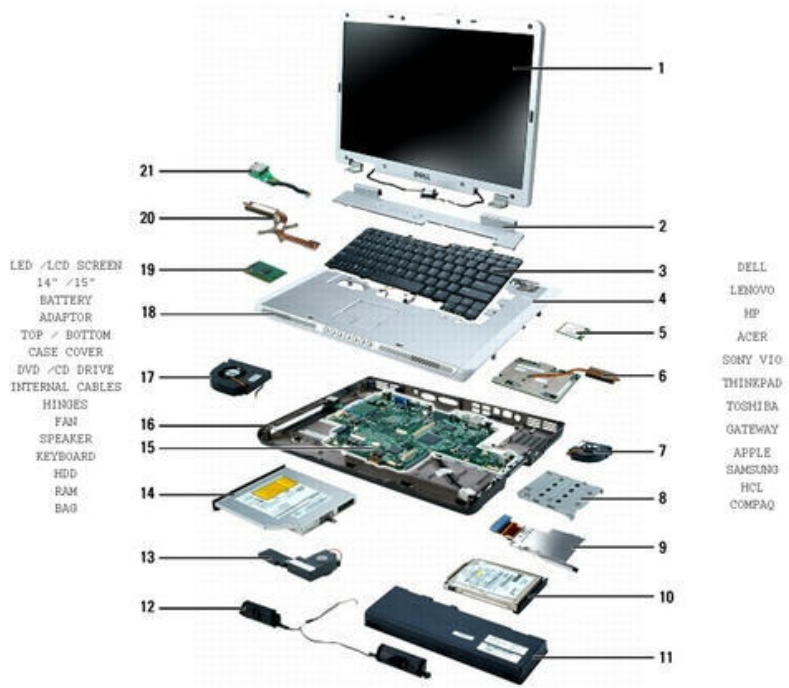
Bioinformatics is a multi-disciplinary field which includes: computer systems management, networking, database design, computer programming, molecular biology

Section 2

Introduction to Computer Science

What is Computer Science?

- **Computer Science** is the study of the foundations of information and computation
- It is the scientific and **practical** approach to computation and its applications

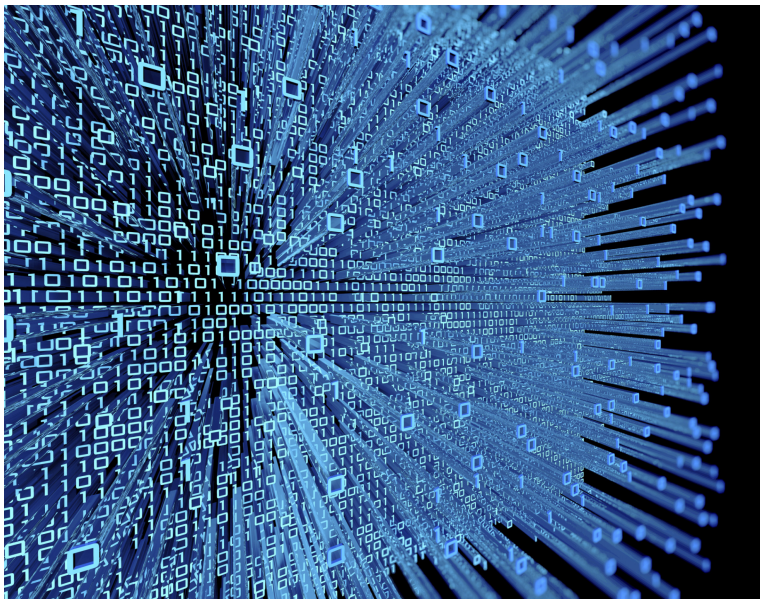


What is a Computer System?

- It is a machine that can transform data into useful information
- It is able to perform **computations** and make **logical decisions**

What is Computation?

- **Computation** is a type of calculation that follows a well defined model
- For example an algorithm (a procedure of well established steps that produce a result)



Types of Computing

Three popular types of computing:

- Personal Computing
  General purpose computer
- Client/Server Computing
  A computer (server) responds to requests across a network to provide some kind of service (web server)
- Cloud Computing
  A shared pool of computing resources accessible over a network

Advantages of Computer Science

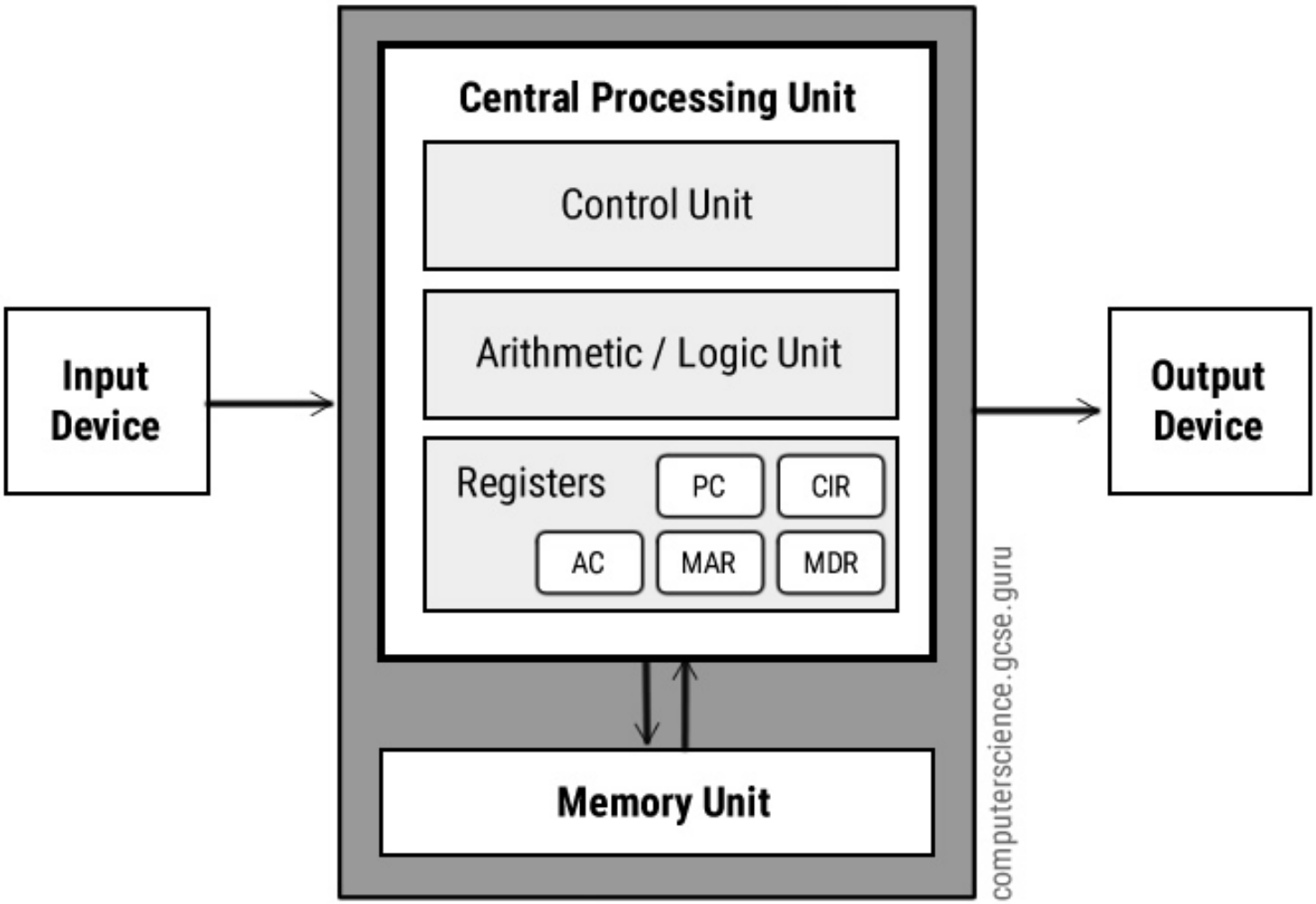
- The **fastest** medium of communication
- Easy solution of arithmetic problems
- Receipt, supply and process of **large volumes** of data at very high speeds

How Computer Work

A computer is an electronic machine that:

- Accepts information and stores it until the information is needed
- Processes the information according to the instructions provided by the user
- Returns the result to the user

How Computer Work



Types of Computer

- Personal computer
- Smartphone
- Mainframe computer
- Super computer
- Cloud computer

Personal computer

– Designed for personal use, as by a person in an office or at home or school

Smartphone

– Small, pocket sized single-user computer based on a microprocessor

Mainframe Computer

- Powerful multi-user computer capable of supporting many hundreds of thousands of users simultaneously

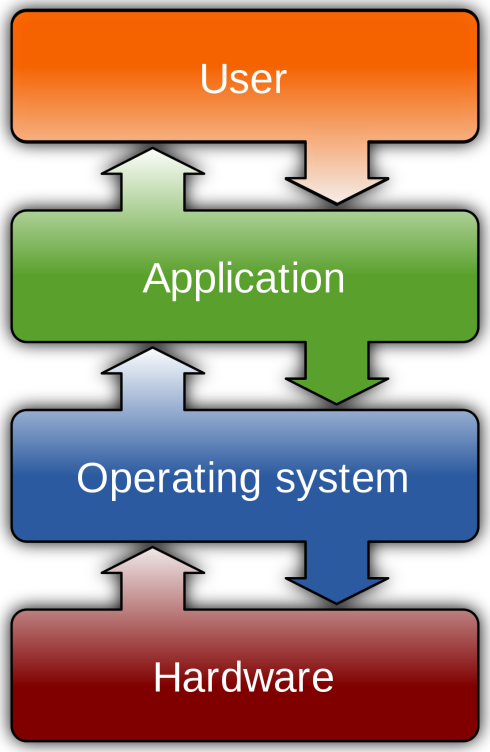
Supercomputer

– An extremely fast computer that can perform hundreds of millions of instructions per second

Hardware and Software

- Hardware includes physical parts of a computer
  For example: CPU, keyboard, mouse, printer, etc
- Software includes an interface between the user and the computer
  For example: Windows, Linux, Mac OS, etc

Operating System



- A software that manages computer hardware
- Constitues the user interface to the computer
- Controls all machine activities nad manages resources

Data representation /1

- A computer is two-state machine (only 1's and 0's are used to represent digits ON and OFF)
- These two states can be easily represented using voltage (1-5 Volt is ON, 0 Volt is OFF)
- 1's and 0's can be used to represent decimal numbers
- A bit, which can assume values 0 or 1, is the smallest unit of memory in a computer

Data representation /2

- 1 bit – 0 or 1 (two different numbers)
- 2 bits – 00, 01, 10, 11 (four numbers)
- 3 bits – 000, 001, 010, 011, 100, 101, 110, 111 (eight numbers)
- ...
- How many numbers with 4, 5, 6, 7, 8 bits?

Data representation /3

Positional value	128	64	32	16	8	4	2	1
Powers of 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Smallest number	0	0	0	0	0	0	0	0
Largest number	1	1	1	1	1	1	1	1

- The decimal equivalent number can be obtained by adding all 2's powers corresponding to 1
- Therefore, the largest number that can be represented with 8 bits is: $128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255$

Data representation /4

Represent 83 in binary

Positional value	128	64	32	16	8	4	2	1
	0							
		1						
			0					
				1				
					0			
						0		
							1	
								1
	0	1	0	1	0	0	1	1

Result: $83 = 64 + 16 + 2 + 1$

Data representation /5

- How to represent characters?
- Answer: Associate a (binary) number to each character
- How many bits?
- We have to store 26 lowercase letters + 26 uppercase letters + 10 figures + punctuations + parentheses + . . . ?
- A byte, 8 bits, can be used to store a character
- The most successful solution is represented by the ASCII (American Standard Code for Information Interchange) format

Data representation /6

Caratt.	Byte	Caratt.	Byte	Caratt.	Byte	Caratt.	Byte
NUL	00000000	SPACE	00100000	@	01000000	'	01100000
SOH	00000001	!	00100001	A	01000001	a	01100001
STH	00000010	"	00100010	B	01000010	b	01100010
EXT	00000011	#	00100011	C	01000011	c	01100011
EQT	00000100	$	00100100	D	01000100	d	01100100
EQN	00000101	%	00100101	E	01000101	e	01100101
ACK	00000110	&	00100110	F	01000110	f	01100110
BEL	00000111	'	00100111	G	01000111	g	01100111
BS	00001000	(	00101000	H	01001000	h	01101000
HT	00001001	)	00101001	I	01001001	i	01101001
LF	00001010	*	00101010	J	01001010	j	01101010
VT	00001011	+	00101011	K	01001011	k	01101011
FF	00001100	,	00101100	L	01001100	l	01101100
CR	00001101	-	00101101	M	01001101	m	01101101
SO	00001110	.	00101110	N	01001110	n	01101110
SI	00001111	/	00101111	O	01001111	o	01101111
DLE	00010000	0	00110000	P	01010000	p	01110000
DC1	00010001	1	00110001	Q	01010001	q	01110001
DC2	00010010	2	00110010	R	01010010	r	01110010
DC3	00010011	3	00110011	S	01010011	s	01110011
DC4	00010100	4	00110100	T	01010100	t	01110100
NAK	00010101	5	00110101	U	01010101	u	01110101
SYN	00010110	6	00110110	V	01010110	v	01110110
ETB	00010111	7	00110111	W	01010111	w	01110111
CAN	00011000	8	00111000	X	01011000	x	01111000
EM	00011001	9	00111001	Y	01011001	y	01111001
SUB	00011010	:	00111010	Z	01011010	z	01111010
ESC	00011011	;	00111011	°	01011011		01111011
FS	00011100	<	00111100	\	01011100	ò	01111100
GS	00011101	=	00111101	é	01011101		01111101
RS	00011110	>	00111110	^	01011110		01111110
US	00011111	?	00111111	_	01011111	DEL	01111111

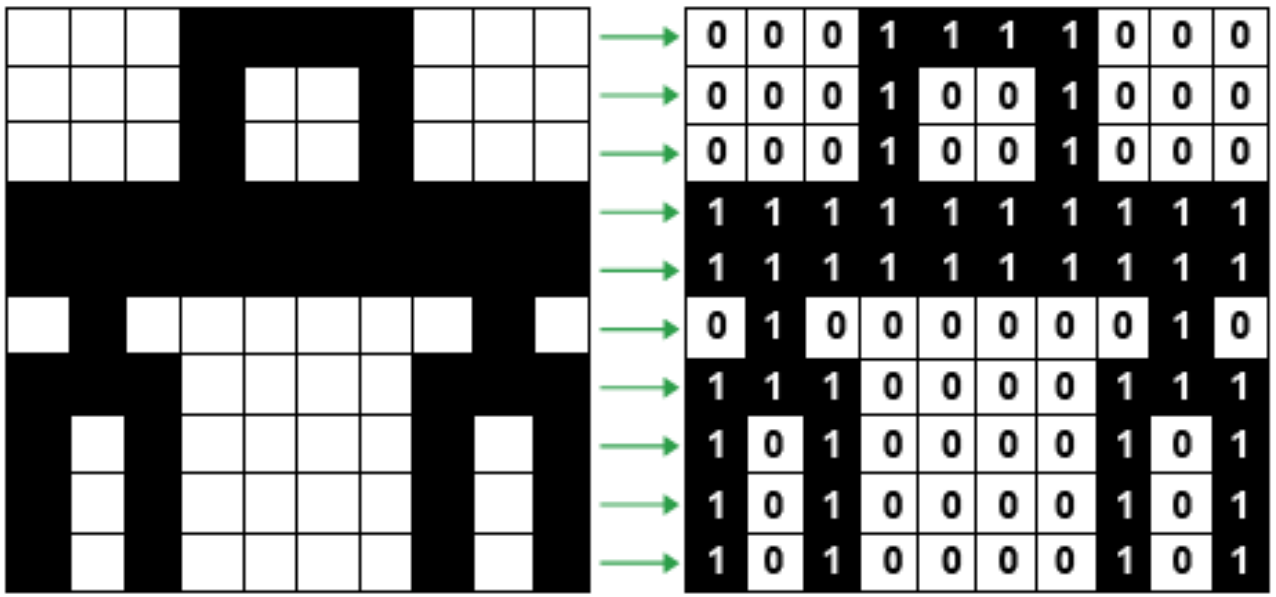
Data representation /7

How to remember memory sizes

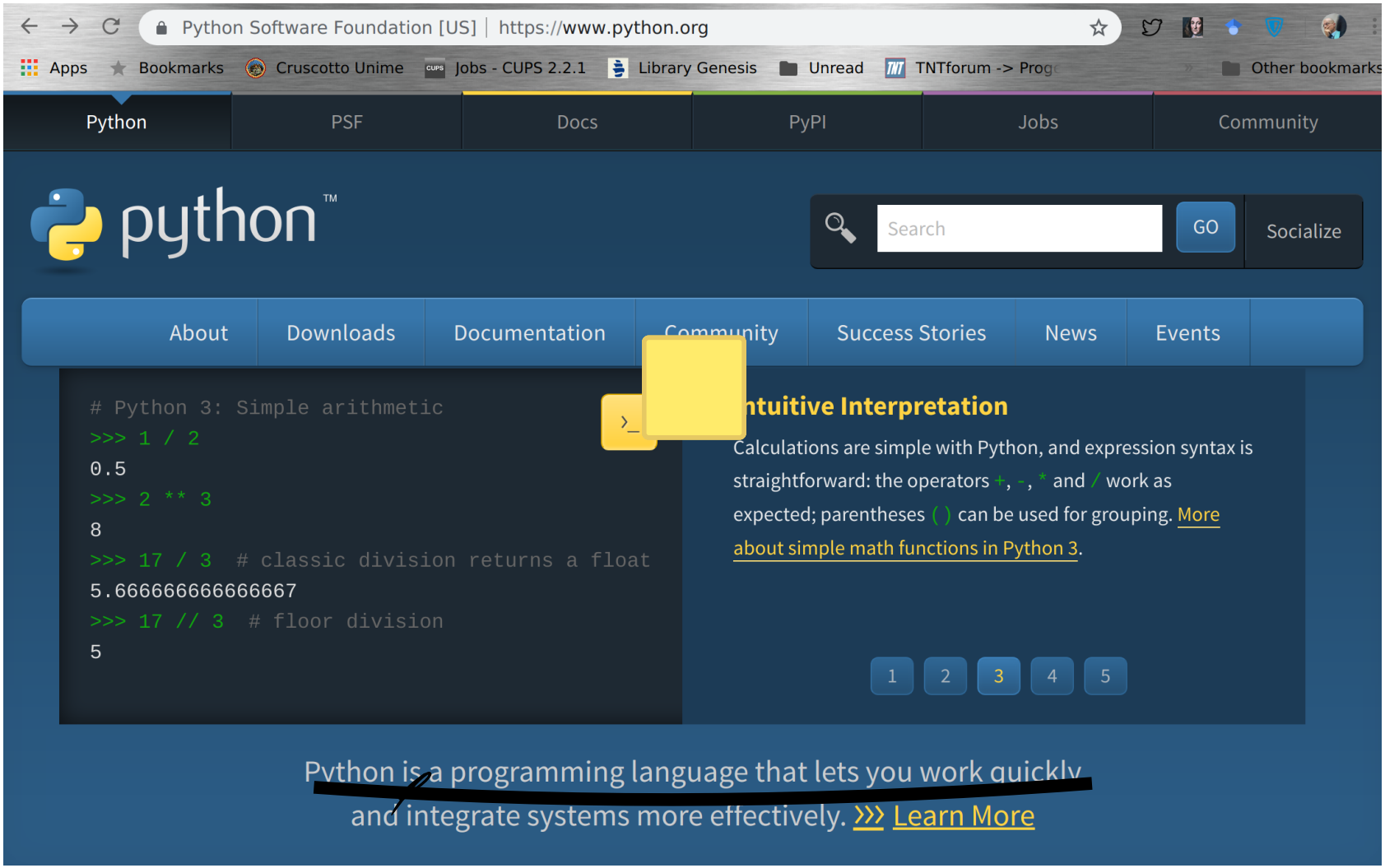
Big	Bit	
Boys	Byte	8 bits
Kicked	kiloByte	1024 Bytes
My	MegaByte	1024 kiloByte
Granny	GigaByte	1024 MegaByte
Twice	TeraByte	1024 GigaByte

Data representation /8

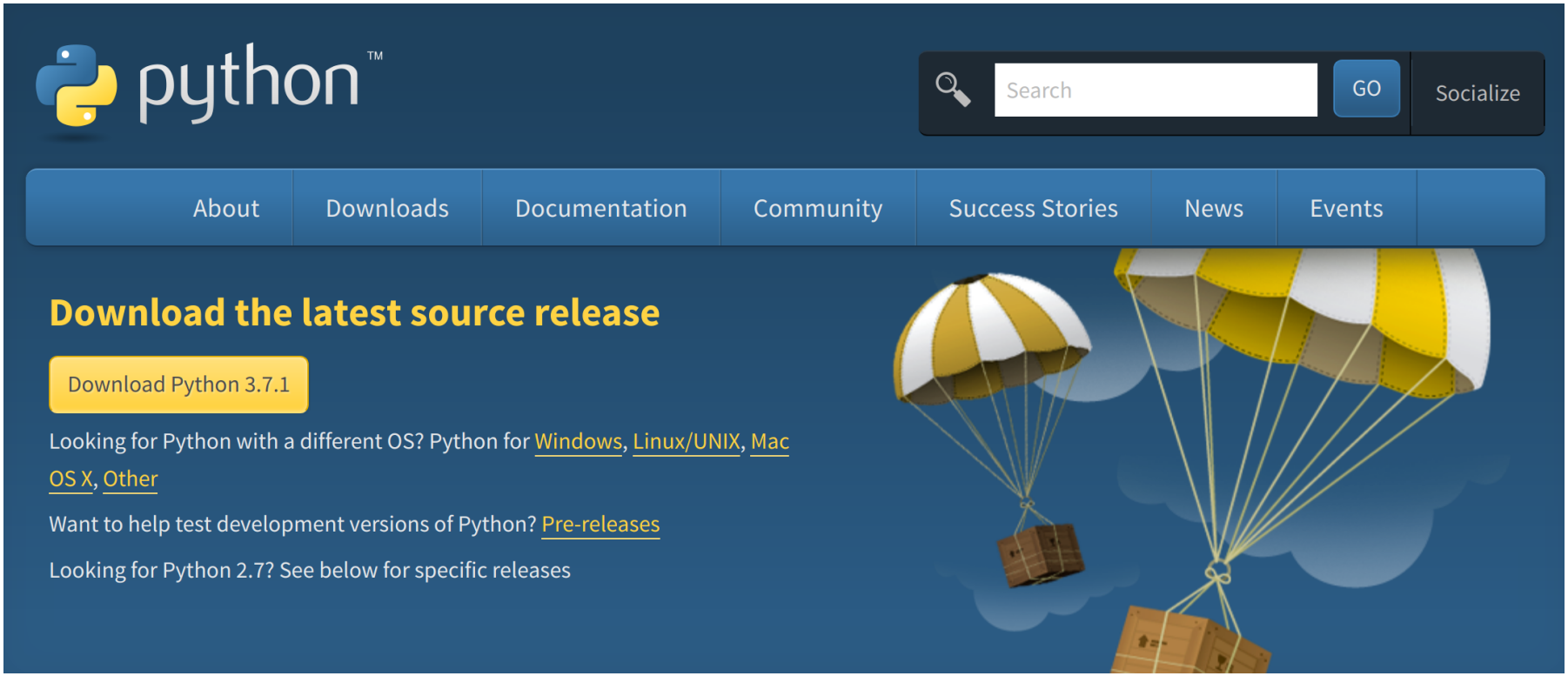
- Graphics are composed of tiny dots called **pixels** each requiring 1 bit
- The picture below requires 100 bits (10×10)



Python


Python Software Foundation [US] | https://www.python.org
Apps
Bookmarks
Cruscotto Unime
Jobs - CUPS 2.2.1
Library Genesis
Unread
TNTforum -> Prog
Other bookmarks
Python
PSF
Docs
PyPI
Jobs
Community
python
Search
GO
Socialize
About
Downloads
Documentation
Community
Success Stories
News
Events
# Python 3: Simple arithmetic
>>> 1 / 2
0.5
>>> 2 ** 3
8
>>> 17 / 3 # classic division returns a float
5.666666666666667
>>> 17 // 3 # floor division
5
ntuitive Interpretation
Calculations are simple with Python, and expression syntax is straightforward: the operators +, -, * and / work as expected; parentheses ( ) can be used for grouping. More about simple math functions in Python 3.
1
2
3
4
5
Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

Python: installation



Python: installation

Python ››› Downloads ››› Windows

Python Releases for Windows

- Latest Python 3 Release - Python 3.7.1
- Latest Python 2 Release - Python 2.7.15

- Python 3.7.1 - 2018-10-20
  - Download Windows x86 web-based installer
  - Download Windows x86 executable installer
  - Download Windows x86 embeddable zip file
  - Download Windows x86-64 web-based installer
  - Download Windows x86-64 executable installer
  - Download Windows x86-64 embeddable zip file
  - Download Windows help file

Python: IDE

One of the most widely used IDE (Integrated Development Environment)

Python: Anaconda

Edit this page on GitHub

Biopython

See also our News feed and Twitter.

Introduction

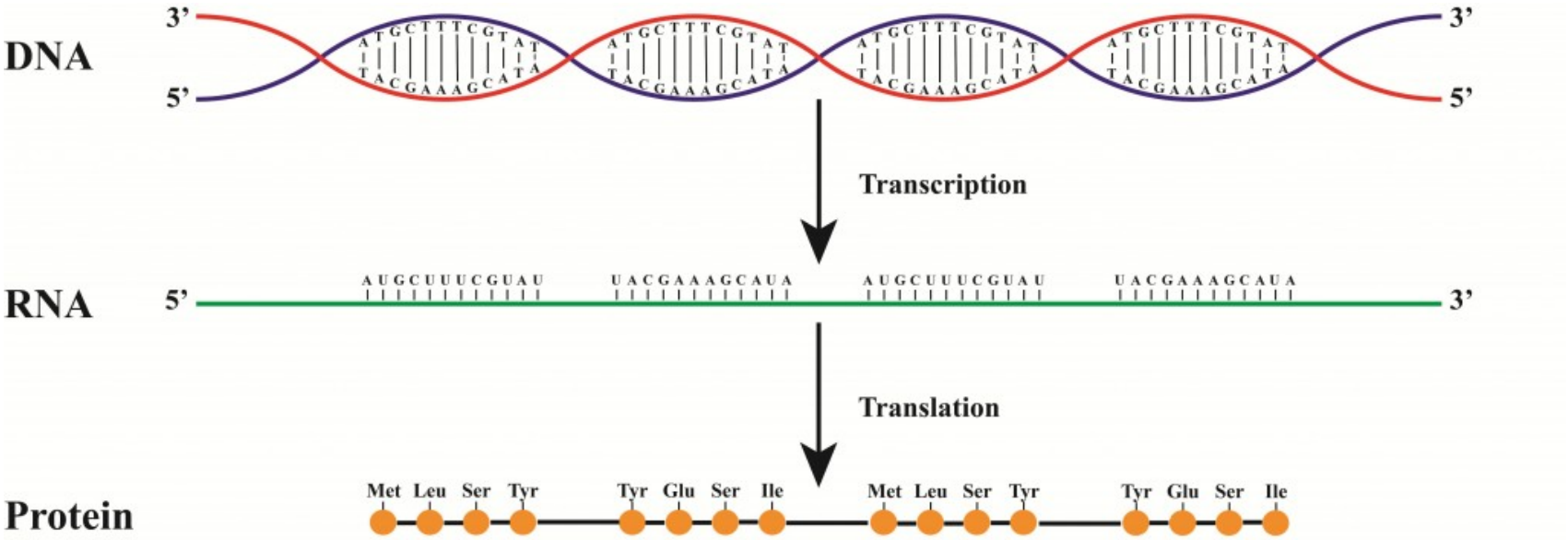
Biopython is a set of freely available tools for biological computation written in Python by an international team of developers.

It is a distributed collaborative effort to develop Python libraries and applications which address the needs of current and future work in bioinformatics. The source code is made available under the Biopython License, which is extremely liberal and compatible with almost every license in the world.

Section 3

Basics of molecular biology

The central dogma



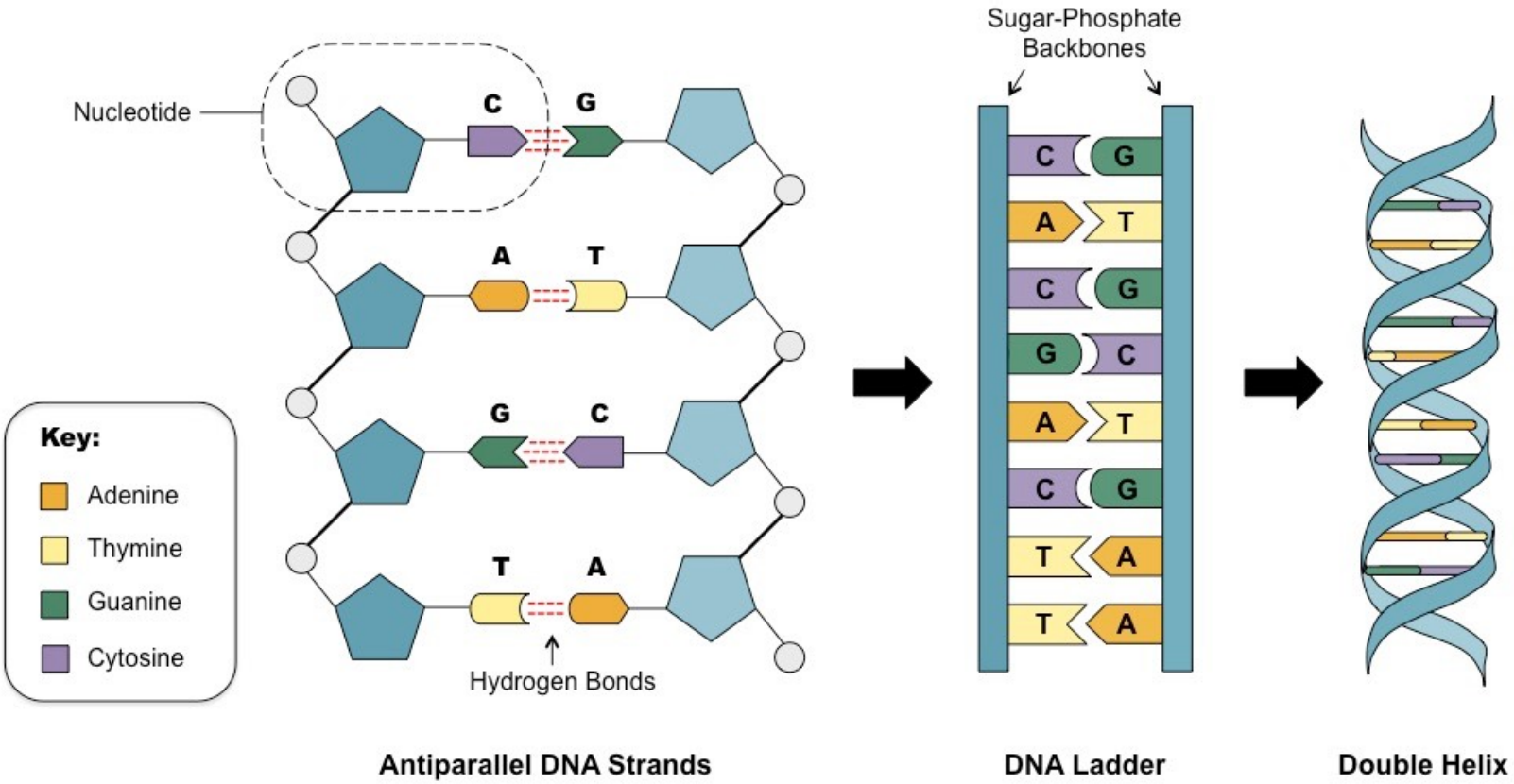
DNA

- Can be imagined as the “recipe ”for an organism
- DNA is composed of small molecules called *nucleotides*: adenine (A), cytosine (C), guanine (G), thymine (T)
- DNA is a polymer, a large molecule consisting of similar units
- a single strand of DNA can be thought of as a string composed of the four letters (A,C,G,T)

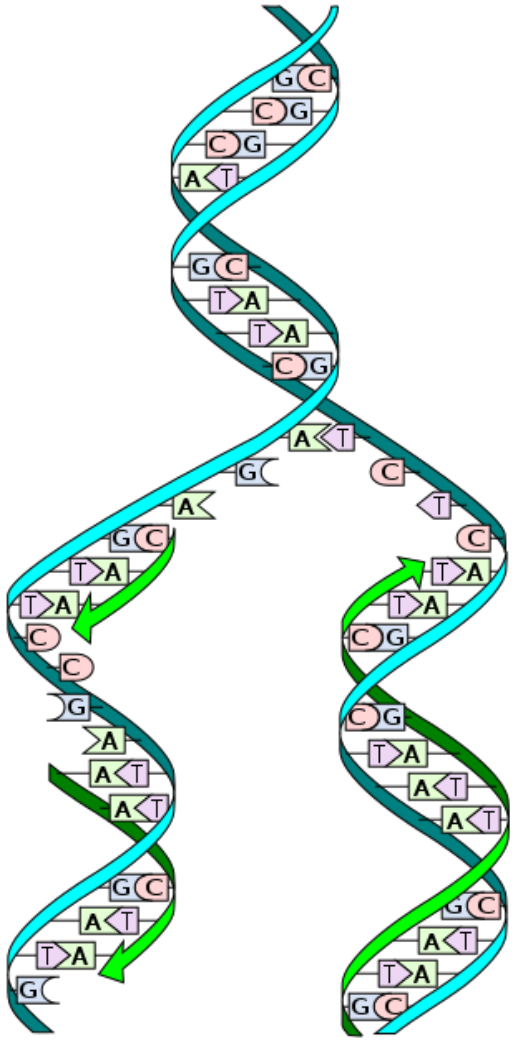
```
ctgctggaccgggtgctaggaccctgactgcccggg
gccgggggtgcggggcccgctgag...
```

The double helix

The double helix



DNA replication



Chromosomes

- DNA is packages into individual chromosomes
- prokaryotes (single-celled organisms lacking nuclei) typically have a large circular chromosome

  Examples: bacteria, archea
- eukaryotes (organisms with nuclei) have a species-specific number of linear chromosomes

  Examples: animals, plants, fungi

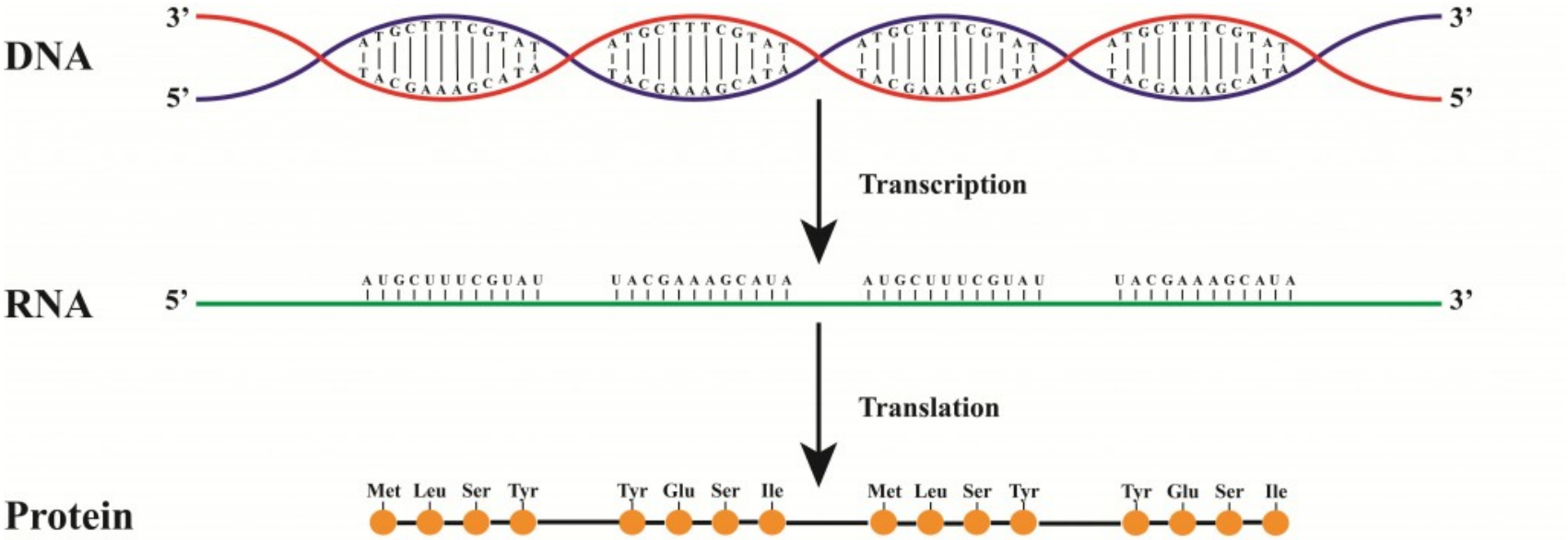
Genomes

- The term **genome** refers to the complete complement of DNA for a given species
- The human genome consists of 23 pairs of chromosomes
- Every cell (with come exceptions) contains the complete genome of an organism

Genes

- **Genes** are the basic units of heredity
- A gene is a sequence of bases carrying the information required for constructing a particular protein
- Such a gene is said to *encode* a protein
- The human genome is composed of $\sim 25,000$ protein-coding genes

The central dogma

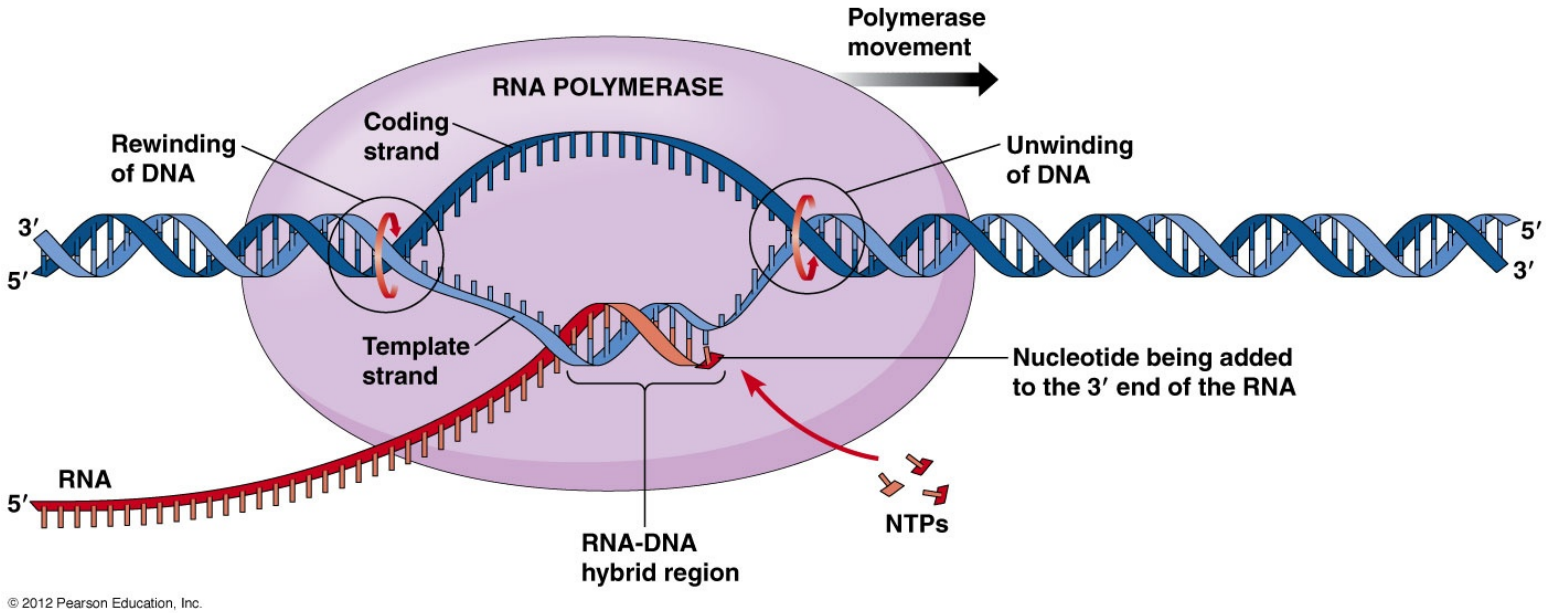


RNA

RNA is like DNA except:

- Backbone is a little different
- Often RNA is single stranded
- The base Uracil (U) is used in place of Thymine (T)
- A strand of RNA can be thought of as a string composed of the four letters: A, C, G, U

Transcription



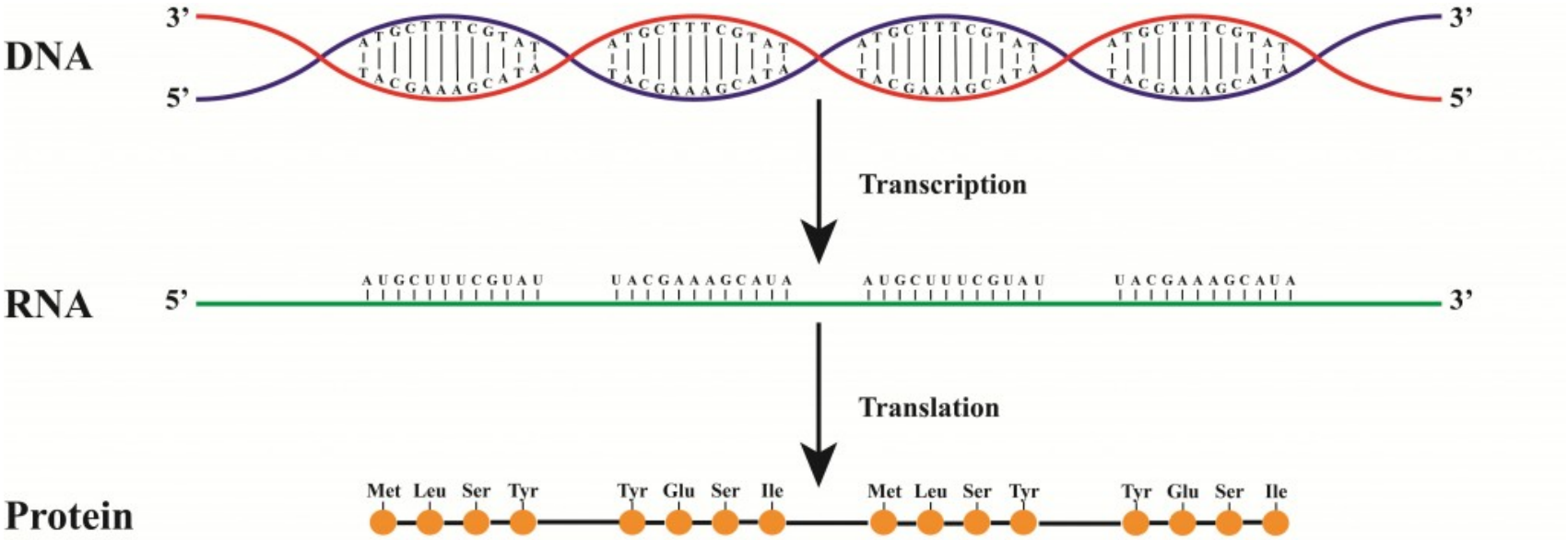
During transcription, a DNA sequence is read by a RNA polymerase, which produces a complementary antiparallel RNA strand (**primary transcript**)

Transcription

The main steps of transcription:

- RNA polymerase binds to promoter DNA (a region of DNA that activates the transcription of a particular gene)
- RNA polymerase creates the so-called **transcription bubble**: the two strands of DNA helix are separated
- RNA polymerase adds RNA nucleotides (complementary to the nucleotides of a DNA strand)
- RNA backbone forms with assistance from RNA polymerase to form an RNA strand

The central dogma



Proteins

- Proteins are molecules composed of polypeptides
- In turn, a polypeptide is a polymer of amino acid monomers
- Cells build their proteins from 20 different amino acids
- A polypeptide can be thought of as a string composed from a 20-character alphabet

Protein functions

- Structural support
- Storage of amino acids
- Transport of other substances
- Coordination of an organism's activities
- Response of cell to chemical stimuli
- Movement
- Protection against disease
- Selective acceleration of chemical reactions

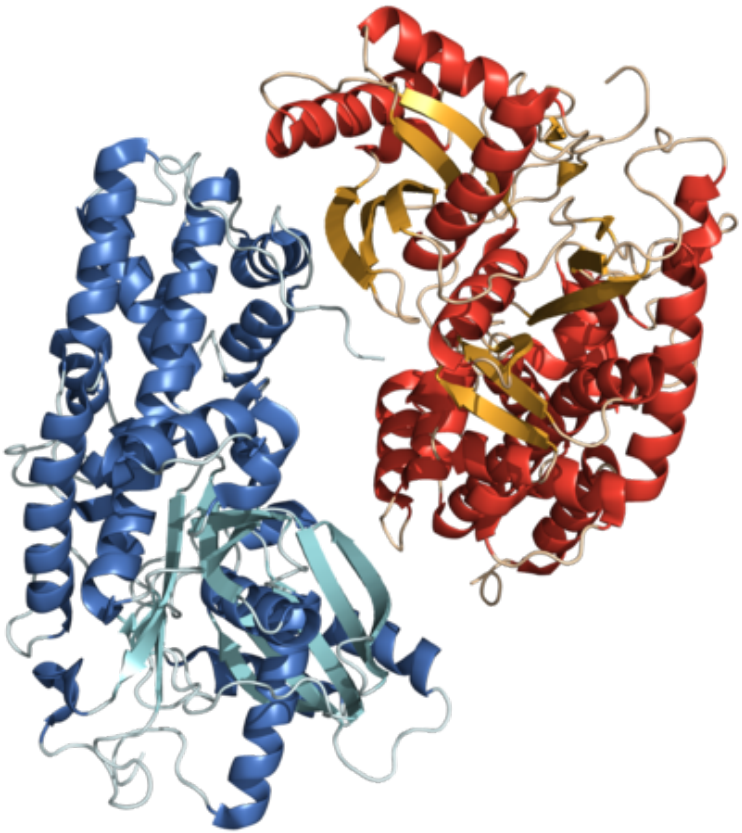
Amino acids

Alanine	Ala	A	Leucine	Leu	L
Arginine	Arg	R	Lysine	Lys	K
Aspartic Acid	Asp	D	Methionine	Met	M
Asparagine	Asn	N	Phenylalanine	Phe	F
Cysteine	Cys	C	Proline	Pro	P
Glutamic Acid	Glu	E	Serine	Ser	S
Glutamine	Gln	Q	Threonine	Thr	T
Glycine	Gly	G	Tryptophan	Trp	W
Histidine	His	H	Tyrosine	Tyr	Y
Isoleucine	Ile	I	Valine	Val	V

Hexokinase: amino acid sequence

MIAAQLLAYYFTELKDDQVKKIDKYLYAMRLSDETLIDIMTRFRKEMKNGLSRDFNPTAT
VKMLPTFVRSIPDGSEKGDFIALDLGGSSFRILRVQVNHEKNQNVHMESEVYDTPENIVH
GSGSQLFDHVAECLGDFMEKRKIKDKKLPVGFTFSFPCQQSKIDEAILITWTKRFKASGV
EGADVVKLLNKAIKKRGDYDANIVAVVNDTVGTMMTCGYDDQHCEVGLIIGTGTNACYME
ELRHIDLVEGDEGRMCINTEWGAFGDDGSLEDIRTEFDREIDRGSLNPGKQLFEKMVSGM
YLGELVRLILVKMAKEGLLFEGRITPELLTRGKFNTSDVSAIEKNKEGLHNAKEILTRLG
VEPSDDDCVSVQHVCTIVSFRSANLVAATLGAILNRLRDNKGTPRLRTTVGVDGSLYKTH
PQYSRRFHKTLRRLVPDSDVRFLLSESGSGKGAAMVTAVAYRLAEQHRQIEETLAHFHLT
KDMLLEVKKRMRAEMELGLRKQTHNNAVVKMLPSFVRRTPDGTENGDFLALDLGGTNFRV
LLVKIRSGKKRTVEMHNKIYAIPIEIMQGTGEELFDHIVSCISDFLDYMGIKGPRMPLGF
TFSFPCQQTSLDAGILITWTKGFKATDCVGHDVVTLLRDAIKRREEFDLDVVAVVNDTVG
TMMTCAYEEPTCEVGLIVGTGSNACYMEEMKNVEMVEGDQGQMCINMEWGAFGDNGCLDD
IRTHYDRLVDEYSLNAGKQRYEKMISGMYLGEIVRNILIDFTKKGFLFRGQISETLKTRG
IFETKFLSQIESDRLALLQVRAILQQLGLNSTCDDSILVKTVCGVVSRRAAQLCGAGMAA
VVDKIRENRGLDRLNVTVGVDGTLYKLHPHFSRIMHQTVKELSPKCNVSFLLSEDGSGKG
AALITAVGVRLRTEASS

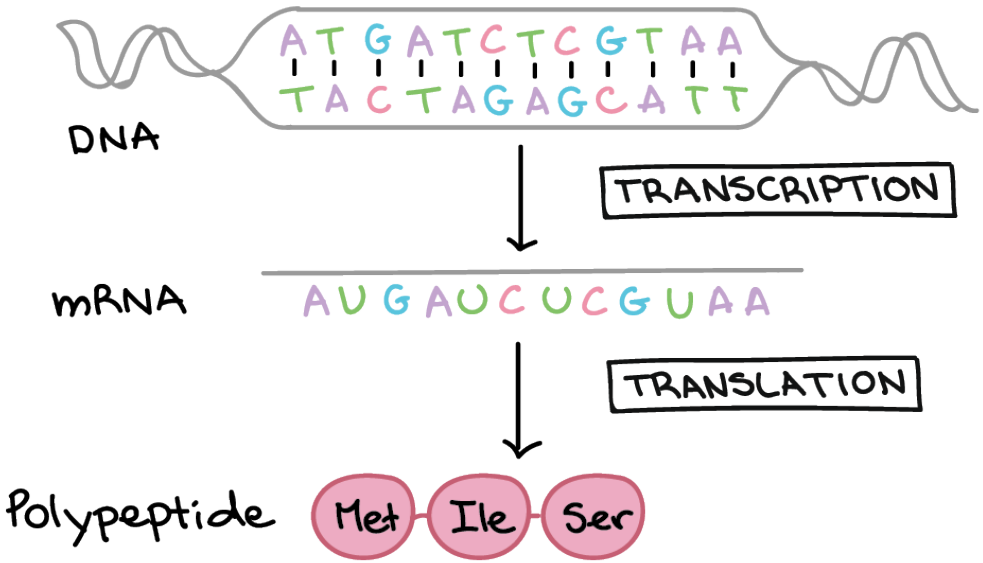
Hexokinase: amino acid sequence



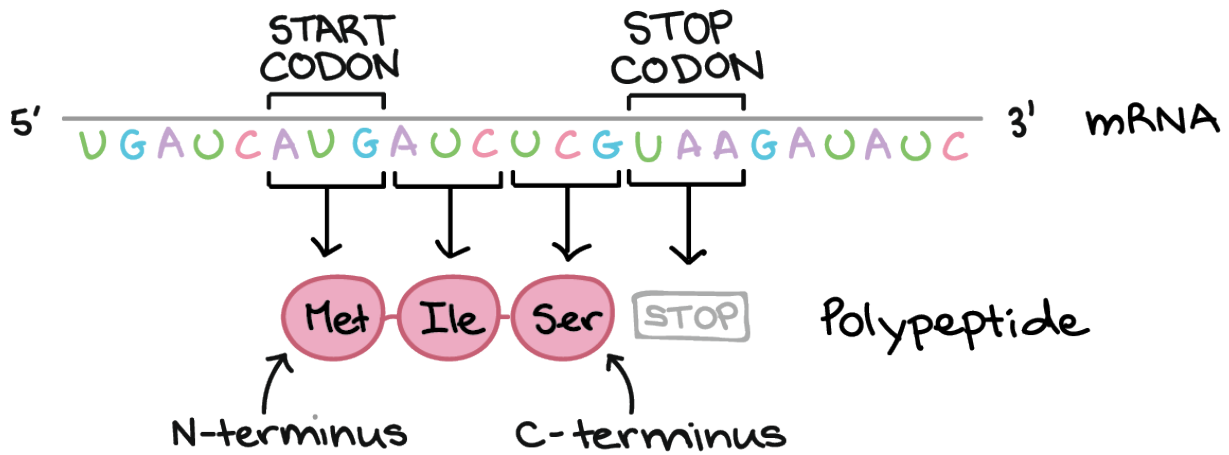
Translation

- **Ribosomes** are in charge of synthesizing proteins from mRNA
- The grouping of codons is called the **reading frame**
- Traslation begins with the **start codon**
- Traslation ends with the **stop codon**

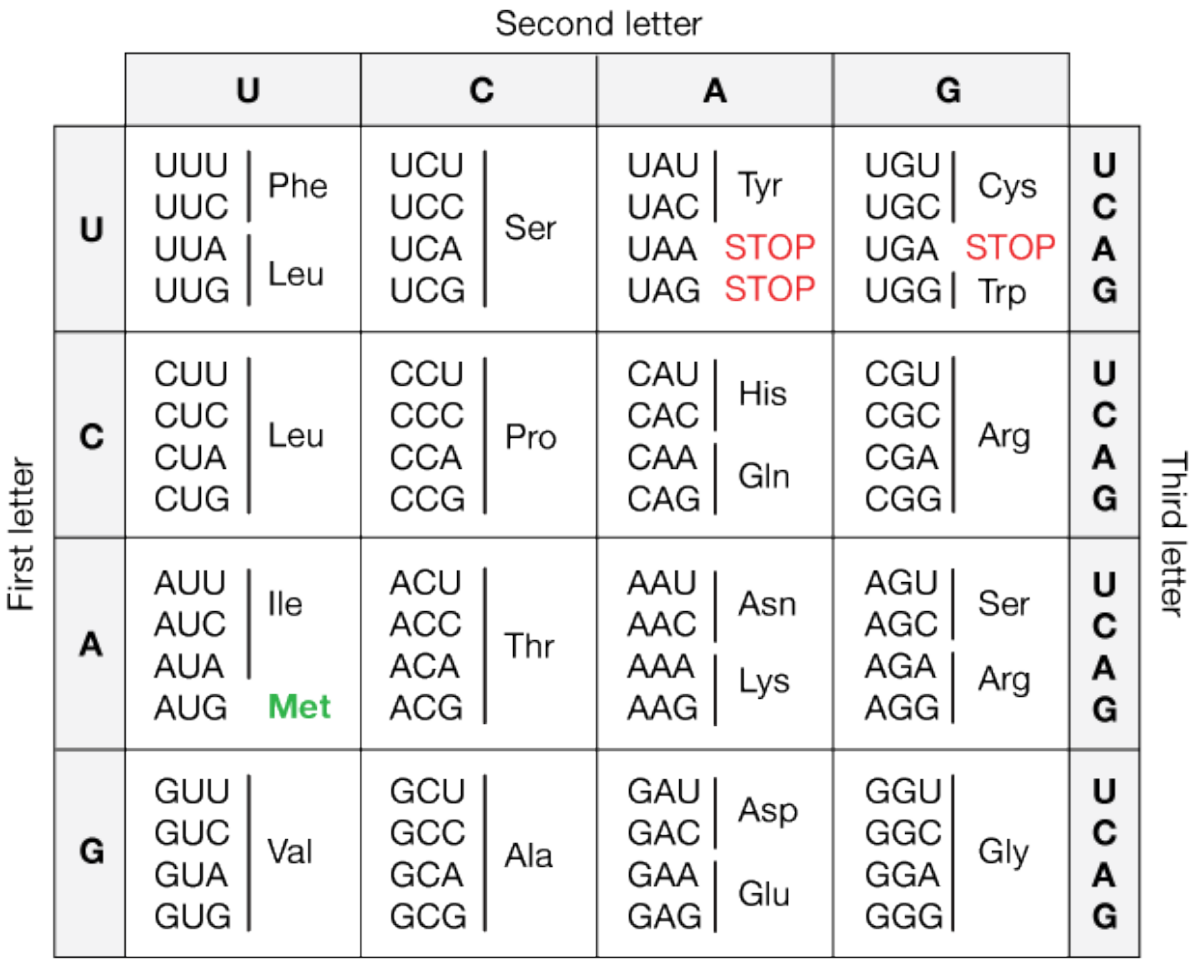
Translation



Translation

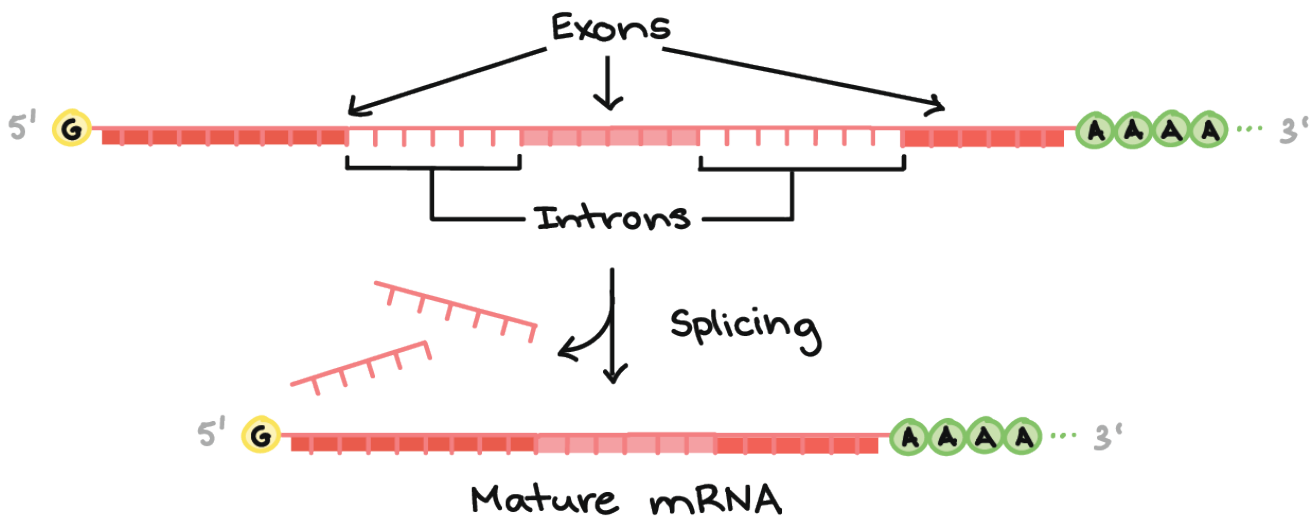


The genetic code



RNA processing in Eukaryotes

- In many Eukaryotes, genes/mRNA consist of alternating **exon/intron** segments
- **Exons** are the coding parts
- **Introns** are spliced out before translation



Section 4

Introduction to Python

Introduction

- Python is a very powerful programming language
- It can perform an extensive variety of tasks
- We shall focus only on biological applications

Comments

- Python exhibits two different kind of comments: single-line and multi-line
- Single-line comments can be inserted into Python code with the # sign
- All text following this sign will be interpreted as a comment (not instructions)
- Multi-line comments must be enclosed between """ and """
- Comments are highly recommended in order to keep notes on what a program is doing or what variables mean
- They are necessary when programs must be read or manipulated by other programmers than the author

Comments

In [1]:
```
a = 5 #This is a single-line comment
```

In [2]:
```
"""
This is a multi-line comment.
It can spread over multiple lines
and must be enclosed by triple quotes
"""
```

Out[2]:
```
'\nThis is a multi-line comment.\nIt can spread over mult
iple lines \nand must be enclosed by triple quotes\n'
```

Assignments

- Python has variables which can adopt numerical or string values
- The variable name can have letters and numbers (a name cannot begin with a number)
- Capital letters are treated as different from lowercase letters
- The value of a variable can be shown by using the `print` function

Assignments

In [1]:
```
a = 5 #This is a single-line comment
```

In [3]:
```
x1 = 'hello'
```

In [4]:
```
y = 12.34
```

In [5]:
```
print(a)
```

```
5
```

In [6]:
```
print(x1,y)
```

```
hello 12.34
```

Numeric data types

Python offers two basic numeric data types:

- `int` (integer) To represent numbers without decimal values
- `float` (floating point number) To represent numbers with decimal values

```
In [1]: a = 12

In [2]: x = 3.45

In [3]: type(a)
Out[3]: int

In [4]: type(x)
Out[4]: float
```

Numeric data types

In [5]:
```
int(x)
```

Out[5]:
```
3
```

In [6]:
```
float(a)
```

Out[6]:
```
12.0
```

In [7]:
```
a = a + 1.3
type(a)
```

Out[7]:
```
float
```

Computations

- Variables can be used in mathematical computations
- Standard math symbols can be used

Function	**Symbol**
Addition	+
Subtraction	−
Multiplication	*
True division	/
Integer division	//
Power	**
Modulus	%

Computations

```
In [1]: a = 5 #This is a single-line comment

In [2]: y = 12.34

In [3]: y // a
Out[3]: 2.0

In [4]: y / a
Out[4]: 2.468
```

Computations

```
In [1]: a = 5 #This is a single-line comment

In [2]: b = 2

In [3]: a**b
Out[3]: 25

In [4]: b**a
Out[4]: 32

In [5]: a % b
Out[5]: 1
```

The `math` module

- Python comes with a math module that contains basic functions
- As it is not immediately available, it must be **imported**
- Import is carried out with the instruction:
  `import math`
- Functions of this module can be invoked prepending to their name the name of the module, *e.g.*,
  `math.sqrt(16)`

The `math` module

```
In [1]: import math

In [2]: math.pow(3,4)
Out[2]: 81.0

In [3]: math.sqrt(45)
Out[3]: 6.708203932499369

In [4]: math.pow(45,0.5)
Out[4]: 6.708203932499369

In [5]: math.pi
Out[5]: 3.141592653589793
```

Python collections

Python offers four methods of collecting items (sequences):

- tuple
- list
- dictionary
- set

Python collections: tuple

- A *tuple* is a collection of items that cannot be changed
- A tuple is encased in round parentheses
- To get a single item from a tuple, square parentheses must be used

```
In [1]: t1 = (10, 3.45, 'hello', 20)

In [2]: print(t1)
        (10, 3.45, 'hello', 20)

In [3]: t1[0]
Out[3]: 10
```

Python collections: tuple

- Tuple items cannot be modified

```
In [1]: t1 = (10, 3.45, 'hello', 20)

In [2]: print(t1)
        (10, 3.45, 'hello', 20)

In [3]: t1[0]
Out[3]: 10

In [5]: t1[0] = 100
        ---------------------------------------------------------------
        TypeError                                 Traceback (most
         recent call last)
        <ipython-input-5-3c8e12ad4afd> in <module>()
        ----> 1 t1[0] = 100
```

Python collections: tuple

- Indices are used to access tuple items (0 is the first element)

```
In [1]: t1 = (10, 3.45, 'hello', 20)

In [2]: print(t1)
        (10, 3.45, 'hello', 20)

In [3]: t1[0]
Out[3]: 10

In [6]: t1[1]
Out[6]: 3.45

In [7]: t1[2]
Out[7]: 'hello'
```

Python collections: tuple

- Trying to access a non-existent item provokes an exception (error)

```
In [8]: t1[3]
Out[8]: 20

In [9]: t1[4]
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-9-93f0b19ad945> in <module>()
----> 1 t1[4]

IndexError: tuple index out of range
```

Python collections: tuple

- Negative indices can be used to access elements starting from the end

```
In [1]: t1 = (10, 3.45, 'hello', 20)

In [10]: t1[-1]
Out[10]: 20

In [11]: t1[-2]
Out[11]: 'hello'
```

Python collections: tuple

– Slicing allows to access more items at once

```
In [1]: t1 = (10, 3.45, 'hello', 20)

In [12]: t1[0:3]
Out[12]: (10, 3.45, 'hello')

In [13]: t1[:3]
Out[13]: (10, 3.45, 'hello')

In [14]: t1[2:]
Out[14]: ('hello', 20)
```

Python collections: list

- A list is similar to a tuple except that it can be altered
- A list can be defined by using square parentheses

```
a = []
b = [10,12.3,'hello']
```

- An item in a list can be replaced
- The number of items in a lista can be increased or decreased

Python collections: list

```
In [16]: l1 = [10, 3.45, 'hello', 20]

In [17]: l1
Out[17]: [10, 3.45, 'hello', 20]

In [18]: l1[0] = 100

In [19]: l1
Out[19]: [100, 3.45, 'hello', 20]
```

Python collections: list

```
In [16]: l1 = [10, 3.45, 'hello', 20]

In [17]: l1
Out[17]: [10, 3.45, 'hello', 20]

In [20]: l1.append(200)

In [22]: l1
Out[22]: [100, 3.45, 'hello', 20, 200]
```

Python collections: list

```
In [28]: l1
Out[28]: ['hello', 20, 200]

In [29]: del l1[0]

In [30]: l1
Out[30]: [20, 200]
```

Python collections: dictionary

- A dictionary relies on the concept of word dictionaries
- Each entry (in Python, **key**) is a word associated to its definition (**value**)
- In Python dictionaries, search can be carried on the keys and not on the values
- A dictionary can be defined by using curly parentheses:

```
a = {}
a['John']=200
```

- The key can be an integer, a float, a tuple or a string

Python collections: dictionary

```
In [32]: a = {}

In [33]: a['x'] = 200

In [34]: a
Out[34]: {'x': 200}

In [35]: a['John'] = 12.34

In [36]: a
Out[36]: {'x': 200, 'John': 12.34}
```

Python collections: dictionary

```
In [37]: a = {}

In [38]: a[10] = 'Jack'

In [40]: a[('John','Doe')] = 35

In [42]: a[100] = [10,20,30]

In [43]: a['Goofy'] = (1,2,3)

In [44]: a
Out[44]: {10: 'Jack', ('John', 'Doe'): 35, 100: [10, 20, 30], 'Goof
         y': (1, 2, 3)}
```

Python collections: set

- A set is like the mathematical sets (an assembly of distinct elements)
- All set mathematical operations can be performed on Python sets

Python collections: set

```
In [46]: a = set((1,2,3,2,5))

In [48]: b = set((1,3,4,5,6,6,4))

In [54]: print(a,b)
{1, 2, 3, 5} {1, 3, 4, 5, 6}

In [51]: a.intersection(b)
Out[51]: {1, 3, 5}

In [52]: b.union(a)
Out[52]: {1, 2, 3, 4, 5, 6}

In [53]: a.difference(b)
Out[53]: {2}
```

Python collections: common operations

```
In [55]: a = [10,20,30,'a','b']

In [56]: len(a)
Out[56]: 5

In [62]: b = {1:'a', 2:'b', 3:'c', 4:[1,2,3]}

In [63]: len(b)
Out[63]: 4

In [64]: b[4]
Out[64]: [1, 2, 3]

In [65]: b[4][0]
Out[65]: 1
```

Python collections: common operations

```
In [55]: a = [10,20,30,'a','b']

In [66]: a.append('hello')

In [68]: a.insert(2,'everybody')

In [69]: a
Out[69]: [10, 20, 'everybody', 30, 'a', 'b', 'hello']

In [70]: a.pop(0)
Out[70]: 10

In [71]: a
Out[71]: [20, 'everybody', 30, 'a', 'b', 'hello']
```

Python collections: strings

- A string is an immutable collection of characters
- Strings can be defined using either single or double quotes
- These two possibilities are useful whenever the strings itself contains quotes
- The extraction of characters from a string can be carried out through slicing

```
In [1]: s1 = 'hello everybody'
```

```
In [1]: s2 = "She told him: 'Stop chatting'. He ..."
```

Python collections: strings

```
In [3]: s1 = 'hello everybody'

In [5]: s1[0]
Out[5]: 'h'

In [6]: s1[:4]
Out[6]: 'hell'

In [7]: s1[5:]
Out[7]: ' everybody'

In [8]: s1[::-1]
Out[8]: 'ydobyreve olleh'
```

Python collections: strings

```
In [9]: s1 = 'hello'

In [10]: s2 = 'everybody'

In [11]: s1+s2
Out[11]: 'helloeverybody'

In [12]: s1+' '+s2
Out[12]: 'hello everybody'

In [13]: s1*3
Out[13]: 'hellohellohello'
```

Python collections: string functions

- Several functions are provided in order to manipulate strings and return information about their contents
- The `find` function finds the location of a substring within a string
- The `rfind` function finds the last location of a substring within a string
- The `count` function counts the number of occurrences of a target string

Python collections: string functions

```
In [14]: s1 = 'this is a string'

In [15]: s1.find('is')
Out[15]: 2

In [16]: s1.rfind('is')
Out[16]: 5

In [17]: s1.count('is')
Out[17]: 2
```

Python collections: string functions

- The `upper` function converts a string into upper case
- The `lower` function converts a string into lower case
- The `split` function splits a string into substrings. Without any arguments the string is split on blank spaces
- The `join` function is the opposite of `split`
- The `replace` function allows to replace a substring with another

Python collections: string functions

```
In [14]: s1 = 'this is a string'

In [18]: s1.upper()
Out[18]: 'THIS IS A STRING'

In [19]: s1
Out[19]: 'this is a string'

In [20]: s2 = s1.upper()

In [21]: s2.lower()
Out[21]: 'this is a string'
```

Python collections: string functions

```
In [14]: s1 = 'this is a string'

In [22]: s1.split()
Out[22]: ['this', 'is', 'a', 'string']

In [23]: s1.split('t')
Out[23]: ['', 'his is a s', 'ring']

In [24]: s1.split('is')
Out[24]: ['th', ' ', ' a string']
```

Python collections: string functions

Example: given a fragment of a DNA strand, deduce the complementary strand

```
In [1]: s1='atgactagcactacgacggactacgacgactacgacgactacagcat
```

```
In [2]: s1
```

```
Out[2]: 'atgactagcactacgacggactacgacgactacgacgactacagcatca
tttattacgactacag'
```

```
In [3]: s2 = s1.replace('a','A')
```

```
In [4]: s2
```

```
Out[4]: 'AtgActAgcActAcgAcggActAcgAcgActAcgAcgActAcAgcAtcA
tttAttAcgActAcAg'
```

Python collections: string functions

```
In [5]: s3 = s2.replace('t','a')

In [6]: s4 = s3.replace('A','t')

In [7]: s5 = s4.replace('c','C')

In [8]: s4

Out[8]: 'tagtcatgctcatcgtcggtcatcgtcgtcatcgtcgtcatctgctact
aaataatcgtcatctg'
```

Python collections: string functions

```
In [9]: s6 = s5.replace('g','c')

In [10]: s7 = s6.replace('C','g')

In [11]: s7

Out[11]: 'tactgatcgtgatgctgcctgatgctgctgatgctgctgatgtcgtagt
aaataatgctgatgtc'
```

Python collections: string functions

```
In [12]: s8 = s7[::-1]

In [13]: s8
Out[13]: 'ctgtagtcgtaataaatgatgctgtagtcgtcgtagtcgtcgtagtccg
tcgtagtgctagtcat'
```

Python collections: string functions

- A more efficient method uses a *lookup table*
- The `maketrans` function creates a lookup table in which each character of the first string is replaced with the first character of the second string

```
In [4]: table = s1.maketrans('acgt','tgca')
```

Python collections: string functions

```
In [4]: table = s1.maketrans('acgt','tgca')

In [5]: comp = s1.translate(table)

In [6]: comp = comp[::-1]

In [7]: comp
Out[7]: 'ctgtagtcgtaataaatgatgctgtagtcgtcgtagtcgtcgtagtccgtc
gtagtgctagtcat'
```

Python Logic Control

- In some situations it is necessary to control the flow of the program
- This amount to take decisions and modify accordingly the program
- In other situations it may be necessary to repeat some instructions
- In Python these operations can be done by using `if`, `while` and `for`

Python Logic Control: the `if` statement

- The `if` command steers the program depending on the truth of a condition
- If, in the example below, the condition ($c \geq 5$) is true the indented instructions will be executed.
- Else the indented instructions will be skipped
- In either case, the flow of the program will continue with the next non indented instruction

In [ ]:
```
if c >= 5:
    instruction 1
    instruction 2
instruction 3
```

Python Logic Control: the `if` statement

In [4]:
```
x = 2
```

In [6]:
```
if x > 3:
    print('Yes: ', x)
    print('Another line')
print('This line will be printed in any case')
```

```
This line will be printed in any case
```

Python Logic Control: the `if` statement

- Indentation is crucial for Python
- Whereever used, the identation must be consistent. It is important that all commands have exactly the same number of spaces
- Python requires 4 spaces (not tabs) for indentation

In [1]:
```
x = 6
```

In [2]:
```
if x > 3:
    print('Yes: ',x)
```

```
Yes:  6
```

Python Logic Control: the `if` statement

Comparison	**Math symbol**	**Keyboard**
Greater than	$>$	`>`
Less than	$<$	`<`
Greater than or equals to	$\geq$	`>=`
Less than or equals to	$\leq$	`<=`
Not equal to	$\neq$	`!=`

Python Logic Control: the `if` statement

- The `else` statement is used to execute instructions if the `if` condition is false

In [4]:
```
x = 2
```

In [7]:
```
if x > 3:
    print('Yes: ', x)
    print('Another line')
else:
    print('No: ', x)
print('This line will be printed in any case')
```

```
No:  2
This line will be printed in any case
```

Python Logic Control: the `if` statement

- The condition for the `if` statement can include multiple tests
- Boolean operators (`and, or, not`) can be used to evaluate two or even more tests
- For example:

```
if x > 3 and y < 2:
if x > 3 or y < 2:
```

Digression: boolean operators

Operand 1	Operand 2	AND
False	False	False
False	True	False
True	False	False
Ture	True	True

Operand 1	Operand 2	OR
False	False	False
False	True	True
True	False	True
Ture	True	True

Operand	NOT
False	True
True	False

Python Logic Control: the `if` statement

In [1]:
```
x = 4
y = 3
```

In [2]:
```
if x > 2 and y < 10:
    print('This is OK')
```

```
This is OK
```

In [3]:
```
if x < 5 and y > 4:
    print('This is OK, too')
else:
    print('Compound condition false')
```

```
Compound condition false
```

Python Logic Control: the `if` statement

- The `elif` statement if equivalent to else-if
- It is an optional condition that is evaluated only if the `if` condition turns out to be false
- The number of `elif` statements is left to the programmer

In [2]:
```
x = 1
```

In [3]:
```
if x > 0:
    print('x is positive')
elif x == 1:
    print('x is equal to 1')
else:
    print('x is negative')
```

```
x is positive
```

Python Logic Control: iterations

- Iterations are used to perform the same instruction repeatedly
- Of course in subsequent iterations there are some differences
- For example, in the first iteration some processing is done over the first element of a string. In the second iteration the **same** processing is done over the second element of the same string
- Two statements are available in Python to implement iterations: `while` and `for`

Python Logic Control: the `while` loop

- The `while` loop performs iteratively the same set of instructions until a condition becomes false
- Avoid infinite loops!
- Infinite loops happen whenever the condition contained in the `while` statement is never updated
- Therefore, the set of instructions contained in the `while` block must be contained at least an instruction in which the condition is modified

Python Logic Control: the `while` loop

In [1]:
```
x = 0
```

In [2]:
```
while x <= 10:
    print ('Value of x = ', x)
    x = x + 1
```

```
Value of x =  0
Value of x =  1
Value of x =  2
Value of x =  3
Value of x =  4
Value of x =  5
Value of x =  6
Value of x =  7
Value of x =  8
Value of x =  9
Value of x =  10
```

Python Logic Control: the `for` loop

- The `for` loop performs iteratively the same set of instructions for a finite number of times
- It is particularly useful when the number of iterations is known in advance
- Iterations can run over the elements of a collection (tuple, list, string, dictionary)
- But they can also run over the indices associated to these elements

Python Logic Control: the `for` loop

In [1]:
```
list1 = ['Jan','Feb','Mar','Apr','May']
```

In [2]:
```
for i in list1:
    print(i)
```

```
Jan
Feb
Mar
Apr
May
```

Python Logic Control: the `for` loop

In [1]:
```
list1 = ['Jan','Feb','Mar','Apr','May']
```

In [3]:
```
len(list1)
```

Out[3]: 5

In [4]:
```
for i in range(len(list1)):
    print(i,"\t",list1[i])
```

```
0        Jan
1        Feb
2        Mar
3        Apr
4        May
```

Python Logic Control: the `range` function

- The `range` function creates a list of integers
- It has three different syntaxes:

Syntax	**Meaning**
`range(stop)`	Integers from 0 to *stop* − 1
`range(start, stop)`	Integers from *start* to *stop* − 1
`range(start, stop, step)`	Integers from *start* to *stop* − 1 skipping *step* elements

Python Logic Control: the `range` function

In [8]:
```
for x in range(3):
    print(x)
```

```
0
1
2
```

In [9]:
```
for x in range(4,10):
    print(x)
```

```
4
5
6
7
8
9
```

Python Logic Control: the `range` function

In [11]:
```
for x in range(0,10,2):
    print(x)
```

```
0
2
4
6
8
```

In [12]:
```
for x in range(10,5,-1):
    print(x)
```

```
10
9
8
7
6
```

Python Logic Control: the `break` and `continue` statements

- Even if the use of the `for` loop is thought for use with a fixed predefined number of iterations, it is possible to alter the regular flow of iterations
- The `break` statement, often in conjunction with a `if` condition, is used to **skip all the remaining iterations**. The flow of the program continues with the first instruction below the `for` block
- The `continue` statement, often in conjunction with a `if` condition, is used to **skip the remaining instructions of the current iteration**. The flow of the program continues with the first instruction of the following iteration

Python Logic Control: the `break` and `continue` statements

In [15]:
```
for i in range(10):
    print(i, end=' ')
    if i > 3:
        break
    print('---')
```

```
0 ---
1 ---
2 ---
3 ---
4
```

Python Logic Control: the `break` and `continue` statements

In [16]:
```
for i in range(10):
    print(i, end=' ')
    if i > 3:
        continue
    print('---')
```

```
0 ---
1 ---
2 ---
3 ---
4 5 6 7 8 9
```

Example 1: the average of a set of random numbers

In [17]:
```
import random
```

In [18]:
```
alist = []
```

In [45]:
```
for i in range(1000):
    r = random.random()
    alist.append(r)
```

In [46]:
```
mean = sum(alist)/len(alist)
```

In [47]:
```
mean
```

Out[47]:
```
0.500030582963378
```

Example 2: the average of a set of random numbers

In [60]:
```
import random
```

In [81]:
```
total = 0
iterations = 1000000000
```

In [82]:
```
for i in range(iterations):
    r = random.random()
    total = total + r
```

In [79]:
```
mean = total/iterations
```

In [80]:
```
mean
```

Out[80]:
```
0.5000131268602146
```

Input/Output: reading a file

- Consider the case in which the text file `mydata.txt` exists on the hard drive
- The goal is to read all its content and store it into some data structure
- A file can be opened for reading using the instruction: `fp=open('filename')`
- The identifier `fp` is a *file pointer*, a bookmark that keeps track of the quantity of information read from the file
- When the file is opened, `fp` is placed at the beginning of the file

Input/Output: reading a file

```
fp = open('mydata.txt')
```

```
data = fp.read()
```

```
fp.close()
```

```
data
```

```
'In the following tutorial you will be guided throug
h the process of installing Jupyter Notebook. Furthe
rmore we'll explore the basic functionality of Jupyt
er Notebook and you'll be able to try out first exam
ples.\n\nThis is at the same time the beginning of a
series of Python-related tutorial on CodingTheSmartW
ay.com. From the very beginning you'll learn everyth
ing to need to know to use Python for scientific com
puting and machine learning use cases.\n\nJupyter No
tebook is a web application that allows you to creat
e and share documents that contain:\n'
```

Input/Output: reading a file

```
fp = open('mydata2.txt')
```

```
for x in fp:
    print(x, end = '')
```

```
line 1
line 2
line 3
line 4
line 5
```

Input/Output: reading a file

```
fp = open('mydata2.txt')
```

```
while True:
    line = fp.readline()
    print(line, end='')
    if not line:
        break
```

```
line 1
line 2
line 3
line 4
line 5
```

Input/Output: reading a file

```
fp = open('mydata2.txt')
```

```
data = fp.readlines()
```

```
print(data)
```

```
['line 1\n', 'line 2\n', 'line 3\n', 'line 4\n', 'li
ne 5\n']
```

Input/Output: reading a file

```
!cat mydata2.txt
```

```
line 1
line 2
line 3
line 4
line 5
```

```
fp = open('mydata2.txt')
```

```
data1 = fp.read(20)
```

```
data1
```

```
'line 1\nline 2\nline 3'
```

Input/Output: writing to a file

- A file can be opened for writing using the instruction:
  `fp=open('filename', 'w')`
- If the file is not empty (or, even better, non existent) its content will be replaced
- Alternatively, appending new data to the end of an already existing file requires:
  `fp=open('filename', 'a')`
- Note that the arguments `'w'` or `'a'` are not compatible with reading some information from that file

Input/Output: writing to a file

```
data = ['Jan','Feb','Mar','Apr','May']
```

```
fp = open('output.txt','w')
```

```
fp.write(str(data))
```

```
35
```

```
fp.close()
```

```
!cat output.txt
```

```
['Jan', 'Feb', 'Mar', 'Apr', 'May']
```

Input/Output: writing to a file

```
data = ['Jan','Feb','Mar','Apr','May']
```

```
fp = open('output.txt','a')
```

```
fp.write(str(data))
```

```
35
```

```
fp.close()
```

```
!cat output.txt
```

```
['Jan', 'Feb', 'Mar', 'Apr', 'May']['Jan', 'Feb', 'M
ar', 'Apr', 'May']
```

Input/Output: `seek`

```
fp = open('mydata.txt')
```

```
data1 = fp.read()
```

```
data1[:10]
```

```
'In the fol'
```

```
data2 = fp.read()
```

```
data2
```

```
''
```

Input/Output: seek

```
fp.seek(0)
```

```
0
```

```
data2 = fp.read()
```

```
data2[:10]
```

```
'In the fol'
```

Input/Output: example

Analyze a DNA string to compute the percentage of Thymine within a sliding window

```
f1 = open('escherichia-coli.fasta')
dna = f1.read()
```

```
dna[:100]
```

```
'CAGTGTCGCCAGCGAAGAAGGAACGGTGAAATCCCCTGGCGAGCACCAG
GTATCTTTGTTGAGCAGTAATATCACGTGATAATGTTTCCTGCCTTTTTC
C'
```

```
tcount1 = dna[:1000].count('ATG')
```

```
tcount1/1000
```

```
0.018
```

Input/Output: example

```
slot = 10
answ = []
for i in range(0,len(dna),slot):
    count = dna[i:i+slot].count('T')
    pct = count/slot
    answ.append(pct)
```

```
sum(answ)/len(answ)
```

```
0.26143141153081495
```

```
len(answ)
```

```
503
```

Python functions

- A **function** is written to contain blocks of instructions that are used repeatedly
- Instead of repeating the same block of instructions several times, the user only needs to invoke the function
- This programming philosophy, known as **modularity**, greatly improves the readability of the code and its reusability

Python functions

A function is composed of the following parts:

- Name
- (optional) Arguments
- (optional, but suggested) Help comments
- Instructions
- (optional) Return statement

Input/Output: example

```
def rect_area(b,h):
    """
    Function to compute the area of a rectangle
    Arguments: the basis and the height
    of the rectangle
    Returns: the area of the rectangle
    """
    a = b * h
    return a
```

```
A = rect_area(10,4)
print(A)
```

```
40
```

Section 5

Biological databases

Introduction

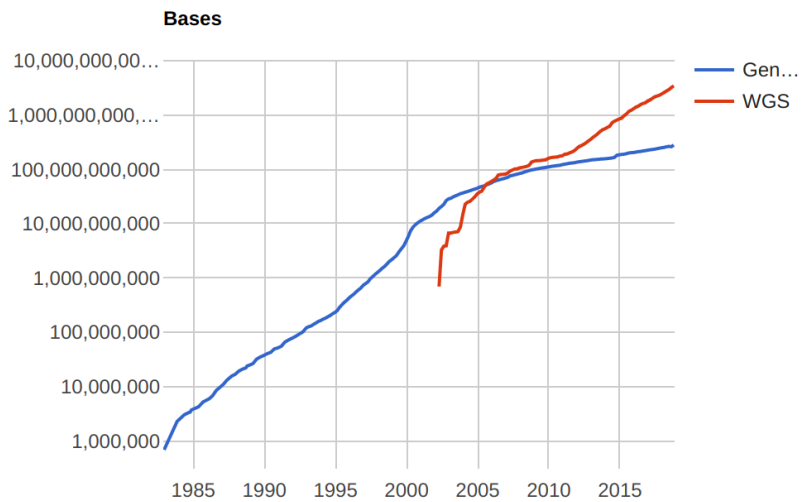
- Applied bioinformatics heavily relies on the collection of sequence data and its associated biological information
- To use these data appropriately, a structured filing system of the data is necessary
- Depending on the kind of data included, different categories of biological databases can be distinguished:
  - **Primary databases** contain primary sequence information (nucleotide or protein)
  - **Secondary databases** summarize the results from analyses of primary protein sequence databases. The goal is to derive common features for sequence classes, which can be used for the classification of unknown sequences

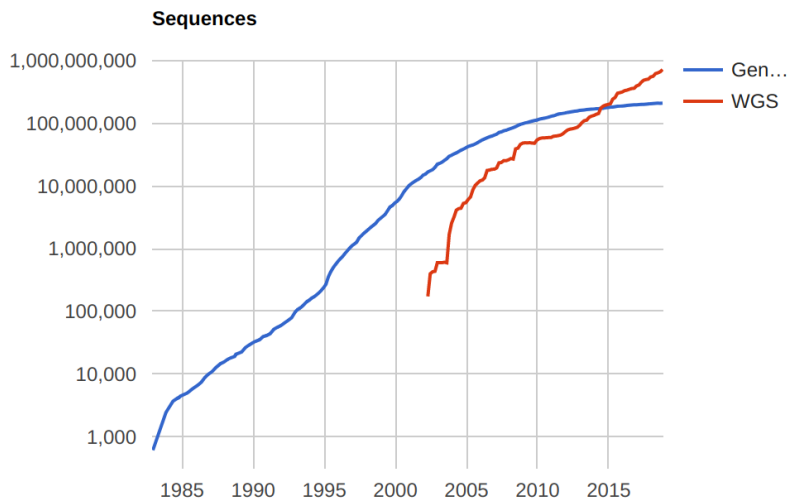
GenBank


https://www.ncbi.nlm.nih.gov/genbank/
Apps
Bookmarks
Cruscotto Unime
Jobs - CUPS 2.2.1
Library Genesis
Unread
TNTforum -> Prog
Other bookmarks
NCBI
Resources
How To
Sign in to NCBI
GenBank
Nucleotide
Search
GenBank
Submit
Genomes
WGS
Metagenomes
TPA
TSA
INSDC
Other
GenBank Overview
What is GenBank?
GenBank ® is the NIH genetic sequence database, an annotated collection of all publicly available DNA sequences (Nucleic Acids Research, 2013 Jan;41(D1):D36-42). GenBank is part of the International Nucleotide Sequence Database Collaboration, which comprises the DNA DataBank of Japan (DDBJ), the European Nucleotide Archive (ENA), and GenBank at NCBI. These three organizations exchange data on a daily basis.
A GenBank release occurs every two months and is available from the ftp site. The release notes for the current version of GenBank provide detailed information about the release and notifications of upcoming changes to GenBank. Release notes for previous GenBank releases are also available. GenBank growth statistics for both the traditional GenBank divisions and the WGS division are available from each release. GenBank growth statistics for both the traditional GenBank divisions and the WGS division are available from each release.
An annotated sample GenBank record for a Saccharomyces cerevisiae gene demonstrates many of the features of the GenBank flat file format.
GenBank Resources
GenBank Home
Submission Types
Submission Tools
Search GenBank
Update GenBank Records

GenBank

GenBank and WGS Statistics





219	Apr 2017	231824951552	200877884	2035032639807	451840147
220	Jun 2017	234997362623	201663568	2164683993369	487891767
221	Aug 2017	240343378258	203180606	2242294609510	499965722
222	Oct 2017	244914705468	203953682	2318156361999	508825331
223	Dec 2017	249722163594	206293625	2466098053327	551063065
224	Feb 2018	253630708098	207040555	2608532210351	564286852
225	Apr 2018	260189141631	208452303	2784740996536	621379029
226	Jun 2018	263957884539	209775348	2944617324086	639804105
227	Aug 2018	260806936411	208831050	3204855013281	665309765
228	Oct 2018	279668290132	209656636	3444172142207	722438528

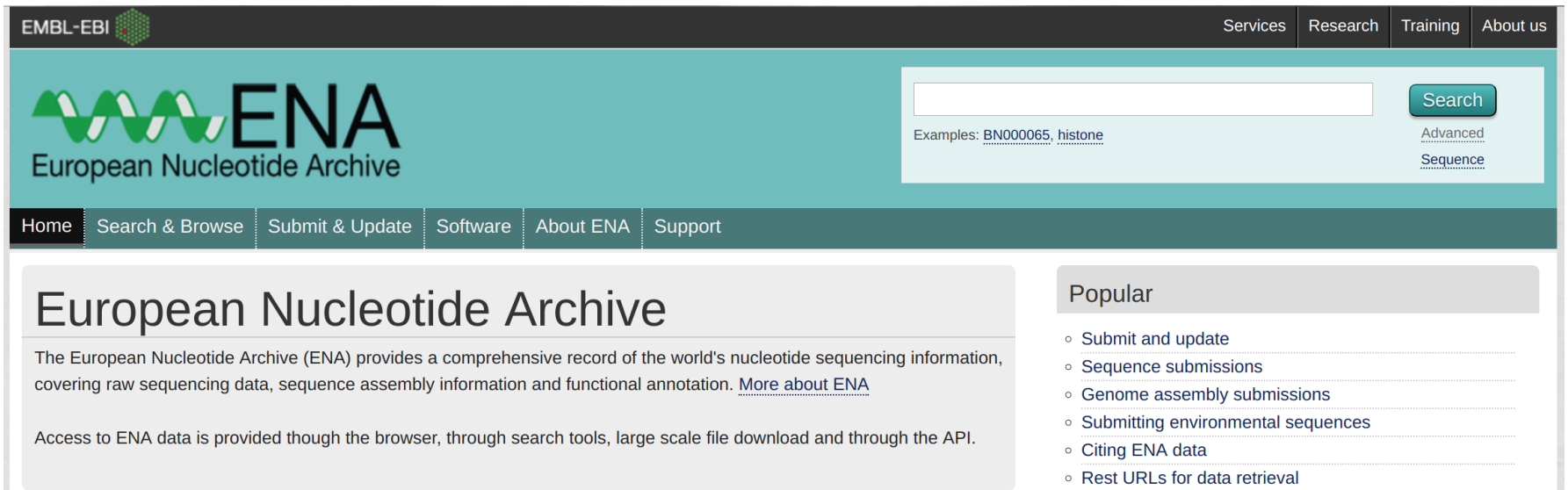
GenBank: an example (Saccharomyces cerevisiae)

```
LOCUS       SCU49845     5028 bp    DNA             PLN       21-JUN-1999
DEFINITION  Saccharomyces cerevisiae TCP1-beta gene, partial cds, and Axl2p
            (AXL2) and Rev7p (REV7) genes, complete cds.
ACCESSION   U49845
VERSION     U49845.1  GI:1293613
KEYWORDS    .
SOURCE      Saccharomyces cerevisiae (baker's yeast)
  ORGANISM  Saccharomyces cerevisiae
            Eukaryota; Fungi; Ascomycota; Saccharomycotina; Saccharomycetes;
            Saccharomycetales; Saccharomycetaceae; Saccharomyces.
REFERENCE   1  (bases 1 to 5028)
  AUTHORS   Torpey,L.E., Gibbs,P.E., Nelson,J. and Lawrence,C.W.
  TITLE     Cloning and sequence of REV7, a gene whose function is required for
            DNA damage-induced mutagenesis in Saccharomyces cerevisiae
  JOURNAL   Yeast 10 (11), 1503-1509 (1994)
  PUBMED    7871890
REFERENCE   2  (bases 1 to 5028)
  AUTHORS   Roemer,T., Madden,K., Chang,J. and Snyder,M.
  TITLE     Selection of axial growth sites in yeast requires Axl2p, a novel
            plasma membrane glycoprotein
  JOURNAL   Genes Dev. 10 (7), 777-793 (1996)
  PUBMED    8846915
REFERENCE   3  (bases 1 to 5028)
  AUTHORS   Roemer,T.
  TITLE     Direct Submission
  JOURNAL   Submitted (22-FEB-1996) Terry Roemer, Biology, Yale University, New
            Haven, CT, USA
FEATURES             Location/Qualifiers
```

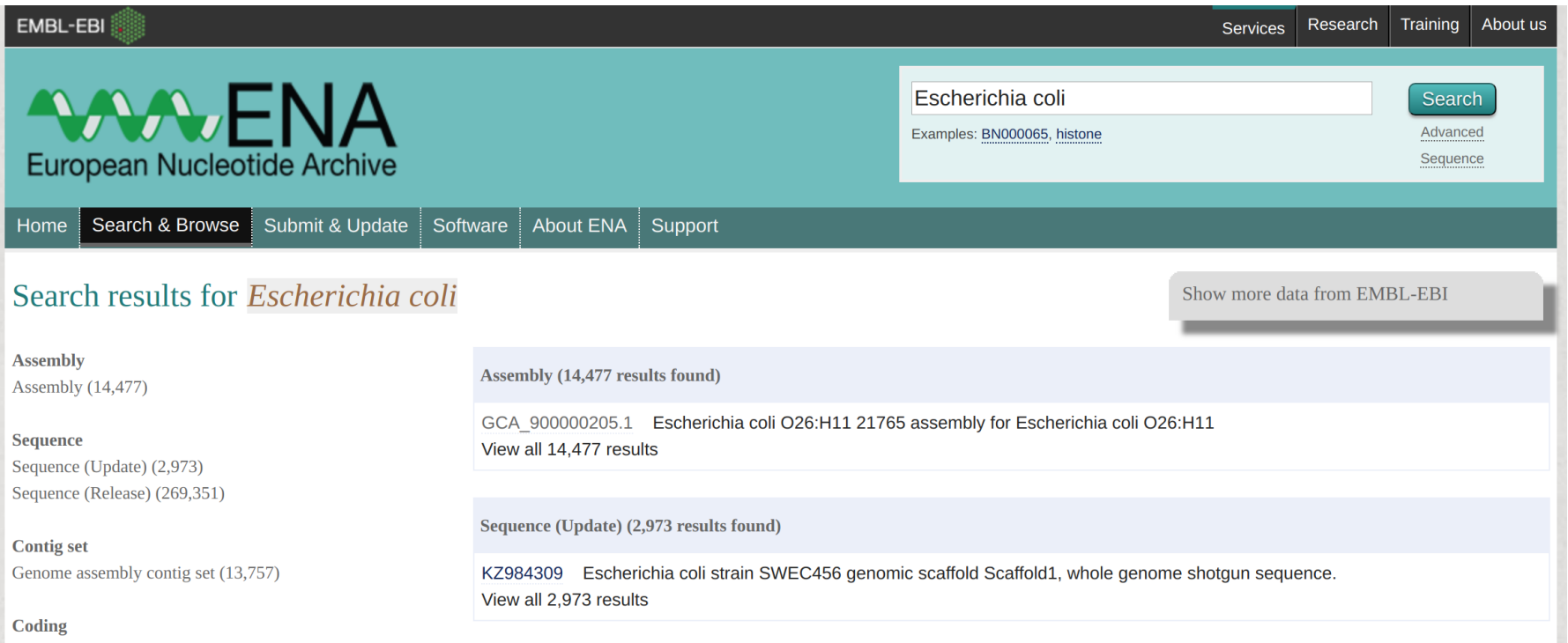
GenBank: an example (Saccharomyces cerevisiae)

```
ORIGIN
        1 gatcctccat atacaacggt atctccacct caggtttaga tctcaacaac ggaaccattg
       61 ccgacatgag acagttaggt atcgtcgaga gttacaagct aaaacgagca gtagtcagct
      121 ctgcatctga agccgctgaa gttctactaa gggtggataa catcatccgt gcaagaccaa
      181 gaaccgccaa tagacaacat atgtaacata tttaggatat acctcgaaaa taataaaccg
      241 ccacactgtc attattataa ttagaaacag aacgcaaaaa ttatccacta tataattcaa
      301 agacgcgaaa aaaaaagaac aacgcgtcat agaacttttg gcaattcgcg tcacaaataa
      361 attttggcaa cttatgtttc ctcttcgagc agtactcgag ccctgtctca agaatgtaat
      421 aatacccatc gtaggtatgg ttaaagatag catctccaca acctcaaagc tccttgccga
      481 gagtcgccct cctttgtcga gtaattttca cttttcatat gagaacttat tttcttattc
      541 tttactctca catcctgtag tgattgacac tgcaacagcc accatcacta gaagaacaga
      601 acaattactt aatagaaaaa ttatatcttc ctcgaaacga tttcctgctt ccaacatcta
      661 cgtatatcaa gaagcattca cttaccatga cacagcttca gatttcatta ttgctgacag
      721 ctactatatc actactccat ctagtagtgg ccacgcccta tgaggcatat cctatcggaa
      781 aacaataccc cccagtggca agagtcaatg aatcgtttac atttcaaatt tccaatgata
      841 cctataaatc gtctgtagac aagacagctc aaataacata caattgcttc gacttaccga
      901 gctggctttc gtttgactct agttctagaa cgttctcagg tgaaccttct tctgacttac
      961 tatctgatgc gaacaccacg ttgtatttca atgtaatact cgagggtacg gactctgccg
     1021 acagcacgtc tttgaacaat acataccaat ttgttgttac aaaccgtcca tccatctcgc
     1081 tatcgtcaga tttcaatcta ttggcgttgt taaaaaacta tggttatact aacggcaaaa
```

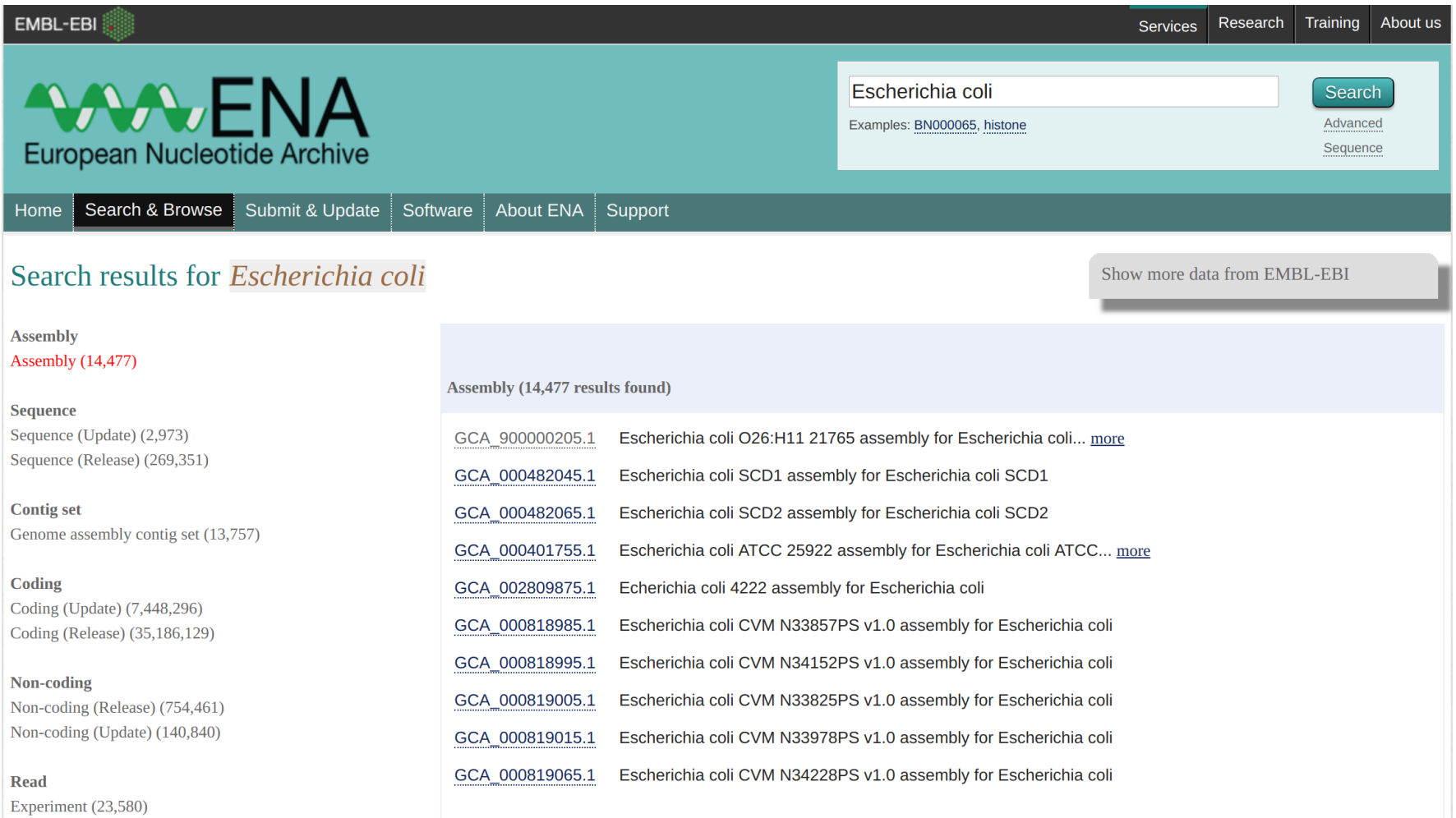
ENA (Europe Nucleotide Archive)



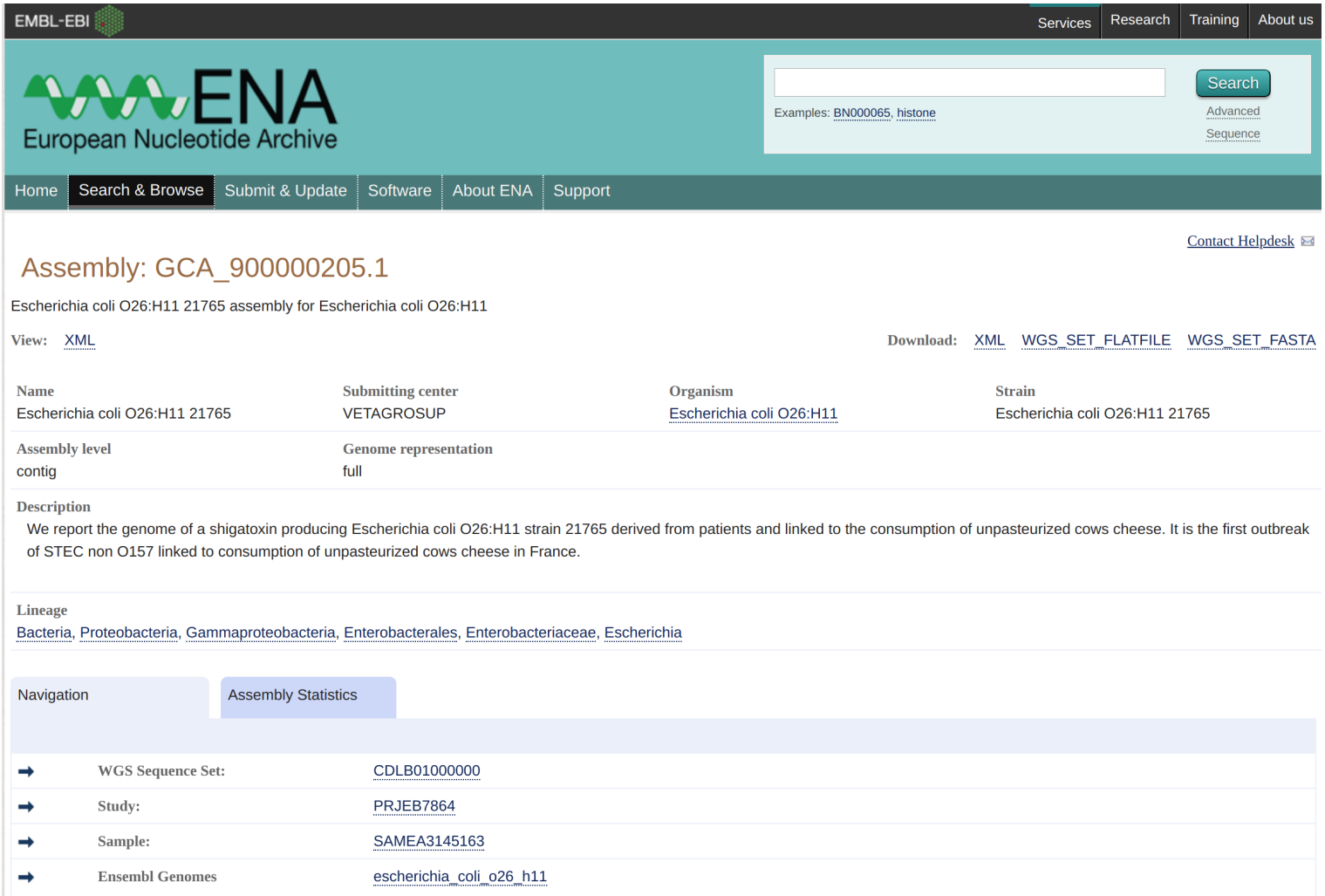
ENA (Europe Nucleotide Archive)



ENA (Europe Nucleotide Archive)



ENA (Europe Nucleotide Archive)



FASTA Format

- FASTA format is a text-based format for representing nucleotide or peptide sequences
- Nucleotides or amino acids are represented using single-letter codes
- The format also allows for sequence names and comments to precede the sequences
- The first line in a FASTA file starts either with a $>$ or ;

FASTA Format

```
>ENA|CDLB01000001|CDLB01000001.1 Escherichia coli O26:H11 strain Escherichia coli
GCGGACTGCCTTTCTCCCAAAGTGATAAACCGGACAGTATCATGGACCGGTTTTCCCGGT
AATCCGTATTTGCAAGGTTGGTTTCACTATGGAACATGAACTTCATTATATCGGTATCGA
CACCGCTAAAGAGAAACTGGATGTCGATGTGTTGCGTCCTGATGGTCGTCATCGCACCAA
AAAATTCGCTAACACCACTAAAGGGCACGATGAGCTGGTGAGCTGGCTGAAAGGTCACAA
GATTGACCATGCGCATATCTGCATCGAAGCGACCGGCACCTATATGGAACCTGTCGCTGA
GTGCCTTTACGATGCTGGCTACATAGTGTCAGTCATTAATCCTGCGCTGGGTAAAGCTTT
CGCTCAGAGTGAAGGACTGCGTAACAAGACTGATACCGTGGATGCGCGCATGCTGGCAGA
GTTCTGTCGTCAGAAGCGCCCTGCAGCCTGGGAAGCGCCTCACCCGCTTGAACGCGCGTT
GCGTGCCCTGGTAGTCCGCCACCAGGCGCTGACAGATATGCACACGCAGGAACTGAATCG
CACTGAAACGGCGCGGGAAGTCCAGAGACCGAGCATTGATGCTCACCTTCTGTGGCTTGA
AGCAGAGCTGAAGCGTCTTGAGAAGCAGATAAAAGACCTGACAGACGATGATCCGGATAT
GAAACACCGCAGGAAACTGCTGGAAAGCATCCCGGGTATCGGAGAGAAAACATCTGCGGT
ATTGCTGGCTTATATCGGTCTGAAGGACCGCTTCGCCCATGCCAGACAGTTCGCCGCTTT
TGCGGGTCTGACACCACGGCGTTATGAATCAGGTAGCAGTGTGAGAGGGGCGAGCCGGAT
```

FASTA Format

```
>ENA|ASJZ01000001|ASJZ01000001.1 Saccharomyces cerevisiae NY1308
CATTACCCTACCTCCCACTCGTTACCCTGTCCTATTCAACCATACCACTCCCAAGCACCA
TCCATCTCTCTACTTACCACCAACCCACCGTCCACGATAACCGTTACCCTCTAATAACCC
ATATCCAACTCCACTACCACTTACCCTACCACTACCCTACCATCCACTATGTCCTACTCA
CTATACTGTTGTTCTACCCACCATGTTGAAACGTTAATAAATGATCGTAAATAATACACA
CATACTTACTCTACCACTCTATACCACCACCACATGCCATACTCACCTTTAATTGTATAC
TGATATGGCATAACGCACACGGATGCTACAGTATATACCATCTCAACTCACCCTACTCTC
ATATTCCACTCCATGACCCATCTCTCACTTCATCAGTACAAATGCACTCACATCATTATG
CACGGCACTTGCCTCAGCGGTCTATACCCTGTGCCATTTACCCATAACGCCCACGATTAT
CCACATTTTAATATCTATATCTCATTCGGCAGCCCCAAATATTGTATAACTGCTCTTAAT
ACATACGTTATACCACTTTTACACCGTATACTAACCACTCAATTTATATACACTTATGCC
AATATTACAAAAAAATCACCACTAAAATCACCTAAACATAAAAATATTCTACTTTTCAAT
AATGATACACATTGGCTTGAAGTATGAACACTATCATGGTATCATTAACTTAAAAGGTCC
TTGATATTGCAATTTGCTTGAACGGATGCCATTTCAGAATATTTCGTACTTATGCAAACC
ATACATTAGAATAATATGCCACCTCACTGTCGTAACACTCTTTATTCACTGAGCTATAAT
```

FASTA Format

```
>sp|P68251|1433B_SHEEP 14-3-3 protein beta/alpha (Fragments) OS=Ovis aries
MTMDKSELVQKAKLAEQAERYDDMAAAMKAVTEQGHELSNEERNLLSVAYKNVVGARRSS
WRVISSIEQKTERNEKKQQMGKEYKMKGDYFRYLSEVASGDNKQTTVSNSQQAYQEAFEI
SKKEMQPTHPIRLGLALNFSVFYYEILNSPEAIAELDTLNEESYKDSTLIMQLLRDNLTL
WTSENQGDEGDAG
```

Section 6

Sequence Comparisons

Introduction

- The comparison of protein and DNA sequences is an important method of applied bioinformatics
- Nature acts conservatively, *i.e.* it does not develop a new kind of biology for every life form. Rather it continuously changes and adapts a proven general concept
- Given this, one may transfer functional information from one protein to another if both possess a certain degree of similarity
- The similarity of two proteins can arise from the evolution from a common ancestor (convergent evolution) ot independently of each other based on different ancestor proteins (divergent evolution)

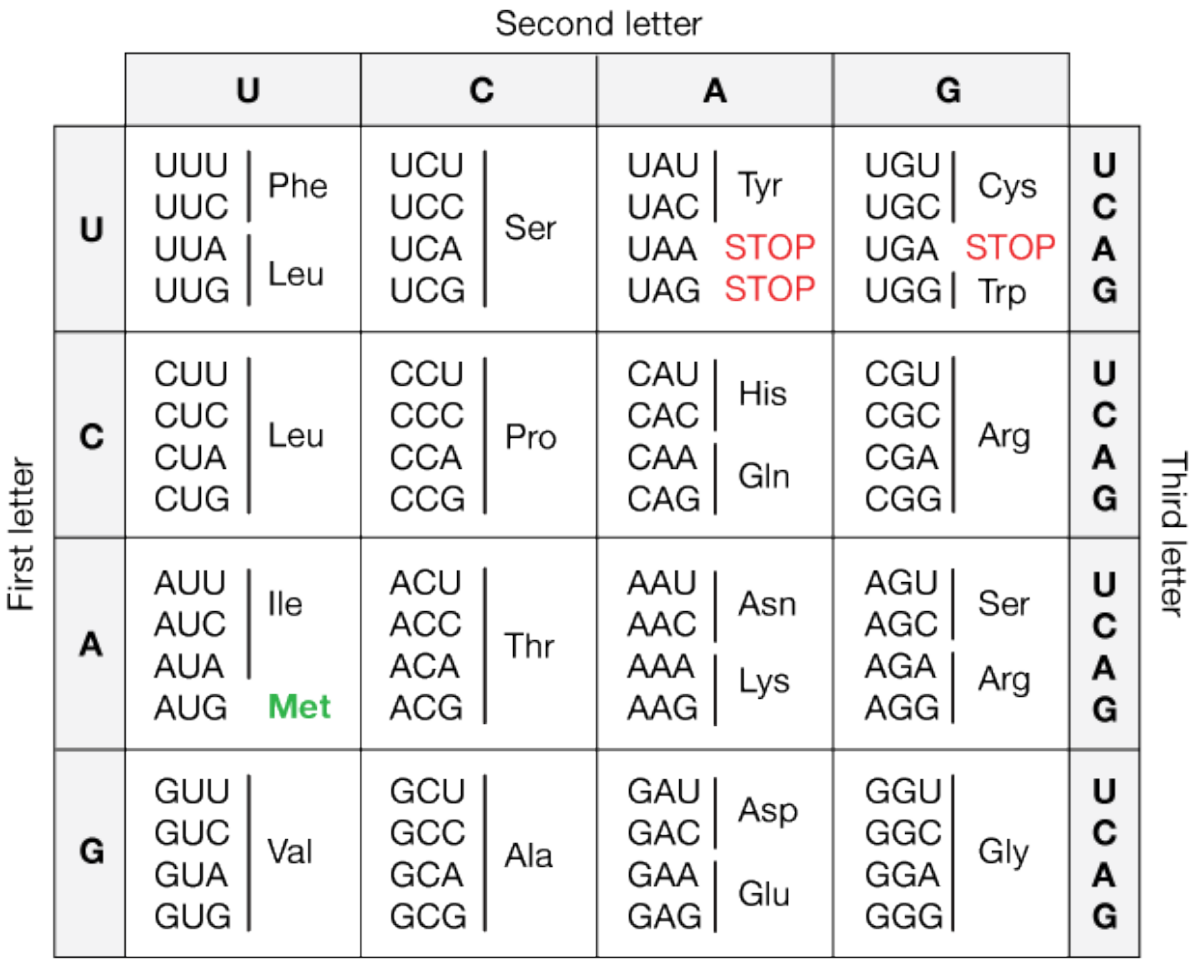
Definitions

- **Homologous:** defines sequences that share evolution history and therefore possess a common ancestral sequence
- **Ortholog:** homologous proteins from different species that possess the same function
- **Paralog:** homologous proteins that have different functions in the same species
- **Identical:** the ratio of the number of identical amino acids or nucleotides in a sequence to the total number of amino acids or nucleotides
- **Similarity matrices:** tables that specify the probability that a sequence will transform into another sequence over time

Amino acids

Alanine	Ala	A	Leucine	Leu	L
Arginine	Arg	R	Lysine	Lys	K
Aspartic Acid	Asp	D	Methionine	Met	M
Asparagine	Asn	N	Phenylalanine	Phe	F
Cysteine	Cys	C	Proline	Pro	P
Glutamic Acid	Glu	E	Serine	Ser	S
Glutamine	Gln	Q	Threonine	Thr	T
Glycine	Gly	G	Tryptophan	Trp	W
Histidine	His	H	Tyrosine	Tyr	Y
Isoleucine	Ile	I	Valine	Val	V

The genetic code



Alignment

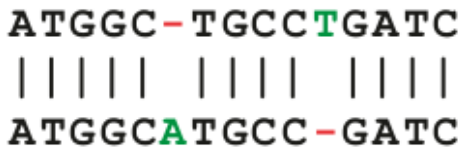

```
ATGGCATGCCTGATC
|||||||||||||||
ATGGCATGCCTGATC
```

Identical sequences

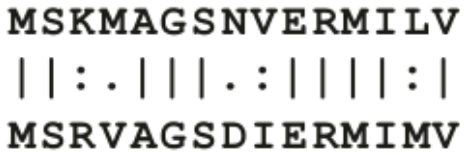
```
MSTPAGSDQERMILV
|||||||||||||||
MSTPAGSDQERMILV
```

Alignment

Match and Mismatch



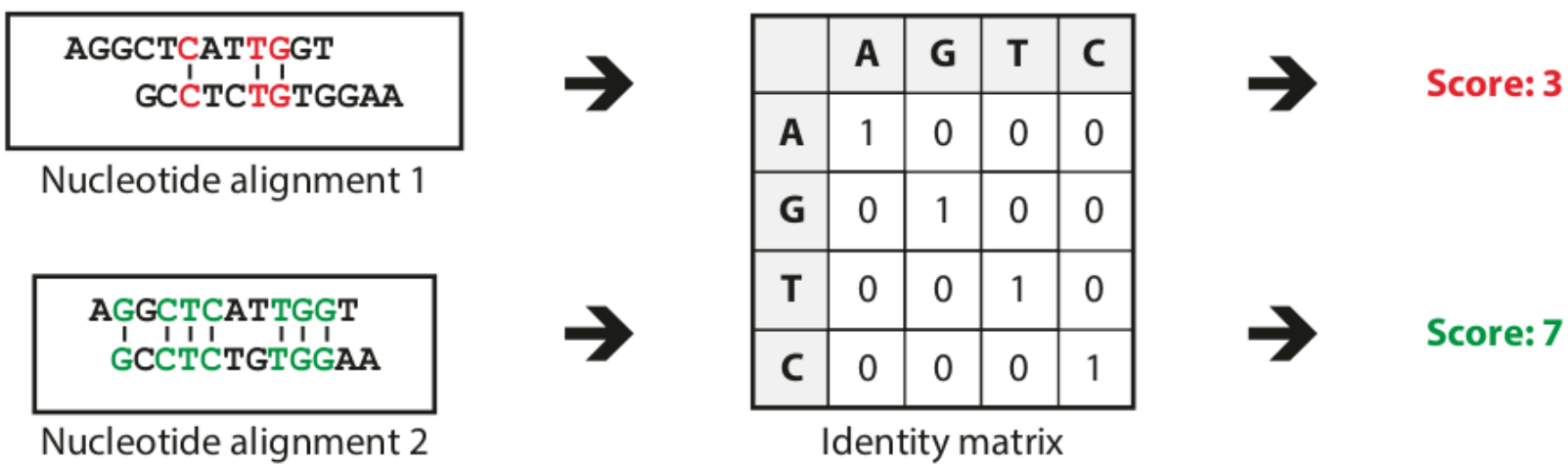
Insertion and Deletion



Conservative mutation

How to calculate the alignment: nucleotides

- Two sequences are arbitrarily placed next to each other and the alignment judged according to the quality measure (the similarity matrix)
- The two sequences are moved relative to one another, and for each position a score is calculated
- The process is repeated until the best alignment is found



How to calculate the alignment: proteins

- In this case, the identity matrix is not sufficient to describe biological and evolutionary processes
- Amino acids are not exchanged with the same probability as might be conceived theoretically
- For example, an exchange of aspartic acid for glutamic acid is frequently observed, as it requires only a mutation of the last nucleotide in the triplet codon (GAT/GAC $\rightarrow$ GAA/GAC)
- For example, an exchange of aspartic acid to tryptophan is rare, as it involves a complete mutation of the whole triplet (GAT/GAC $\rightarrow$ TGG)

How to calculate the alignment: proteins

- The mutation of aspartic acid into glutamic acid occurs more often because they have similar properties
- Instead, aspartic acid and tryptophan are chemically different
- The aspartic acid is hydrophilic and is generally found at the surface of proteins
- Tryptophan is hydrophobic and is found in the center of proteins
- These mutations, accompanied by striking changes of functionality rarely happen

How to calculate the alignment: proteins

- Most algorithms use substitution matrices to align protein sequences
- These matrices describe the probability that amino acids will be exchanged during the course of evolution
- More precisely, positive values suggest an evolution which may lead to an exchange
- Negative values, instead, suggest a coincidental occurrence

How to calculate the alignment: proteins

```
GMIDLITARCAYPSWTGH
  IEVRTAKCAYPGWSGHY
```

Amino acid alignment 1

```
GMIDLITARCAYPSWTGH
      IEVRTAKCAYPGWSGHY
```

Amino acid alignment 1

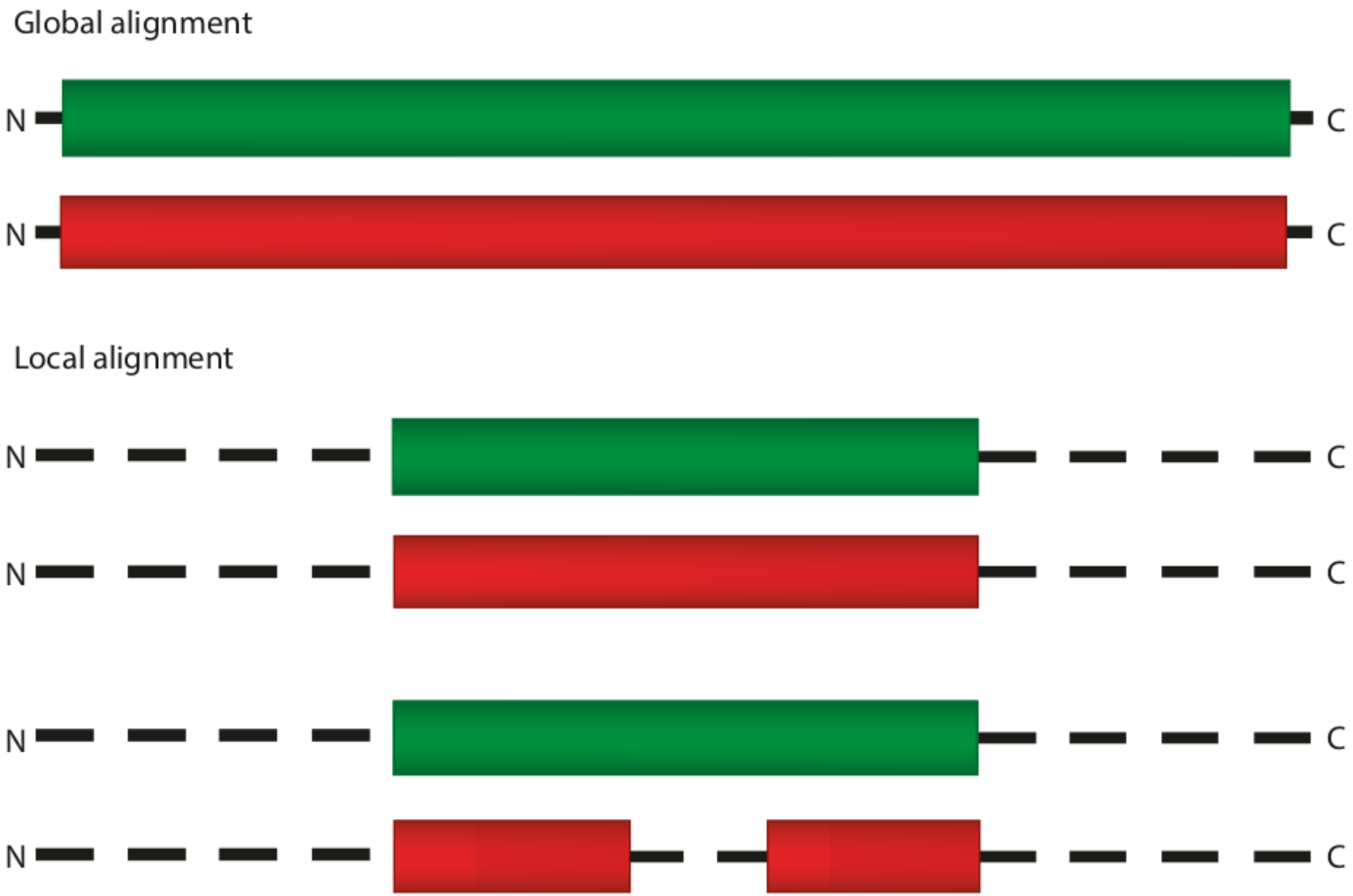
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	**9**	-1	-1	-3	0	-3	-3	-3	-4	-3	-3	-3	-3	-1	-1	-1	-1	-2	-2	-2
S	-1	**4**	1	-1	1	0	1	0	0	0	-1	-1	0	-1	-2	-2	-2	-2	-2	-3
T	-1	1	**4**	1	-1	1	0	1	0	0	0	-1	0	-1	-2	-2	-2	-2	-2	-3
P	-3	-1	1	**7**	-1	-2	-1	-1	-1	-1	-2	-2	-1	-2	-3	-3	-2	-4	-3	-4
A	0	1	-1	-1	**4**	0	-1	-2	-1	-1	-2	-1	-1	-1	-1	-1	-2	-2	-2	-3
G	-3	0	1	-2	0	**6**	-2	-1	-2	-2	-2	-2	-2	-3	-4	-4	0	-3	-3	-2
N	-3	1	0	-2	-2	0	**6**	1	0	0	-1	0	0	-2	-3	-3	-3	-3	-2	-4
D	-3	0	1	-1	-2	-1	1	**6**	2	0	-1	-2	-1	-3	-3	-4	-3	-3	-3	-4
E	-4	0	0	-1	-1	-2	0	2	**5**	2	0	0	1	-2	-3	-3	-3	-3	-2	-3
Q	-3	0	0	-1	-1	-2	0	0	2	**5**	0	1	1	0	-3	-2	-2	-3	-1	-2
H	-3	-1	0	-2	-2	-2	1	1	0	0	**8**	0	-1	-2	-3	-3	-2	-1	2	-2
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	**5**	2	-1	-3	-2	-3	-3	-2	-3
K	-3	0	0	1	-1	-2	0	-1	1	1	-1	2	**5**	-1	-3	-2	-3	-3	-2	-3
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	**5**	1	2	-2	0	-1	-1
I	-1	-2	-2	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	**4**	2	1	0	-1	-3
L	-1	-2	-2	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	**4**	3	0	-1	-2
V	-1	-2	-2	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	**4**	-1	-1	-3
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	**6**	3	1
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	-2	-2	-2	-1	-1	-1	-1	3	**7**	2
W	-2	-3	-3	-4	-3	-2	-4	-4	-3	-3	-2	-3	-3	-1	-3	-2	-3	1	2	**11**

BLOSUM62 matrix

Score: 65

Score: 19

How to calculate the alignment: proteins



Global alignment

- **Global alignment** requires that complete nucleotide or protein sequences are compared to one another over the entire sequence
- Even very similar sequences can have single deletions or insertions and, consequently, a different number of amino acids or nucleotides. In such cases, gaps must be inserted into the sequences
- To prevent that all possible sequences can be aligned by the introduction of gaps, scoring penalties are given for the introduction of gaps and their extension
- The alignment with the highest score is considered the optimal sequence comparison

Global alignment

	M	T	P	A	R	G	S	A	L	S
M	**5**	-1	-2	-1	-1	-3	-1	-1	2	-1
T	-1	**5**	-1	0	-1	-2	-1	0	-1	-1
P	-2	-1	**7**	-1	-2	-2	-1	-1	-3	-1
V	1	0	-2	**0**	-3	-3	-2	0	-1	-2
R	-1	-1	-2	-1	**5**	-2	-1	-1	-2	-1
R	-1	-1	-2	-1	5	**-2**	-1	-1	-2	-1
S	-1	1	-1	1	-1	0	**4**	1	-2	4
L	2	-1	-3	-1	-2	-4	-2	-1	**4**	-2
S	-1	-1	-1	-1	-1	0	4	1	-2	**4**

	M	T	P	A	R	G	S	A	L	S
M	**32**									
T		**27**								
P			**22**							
V				**15**						
R					**15**					
R						**10**				
S							**12**			
L								**8**	**8**	
S										**4**

```
c
1 MTPARGSALS 10          Length: 10
  |||.|.| ||            BlOSUM62, Gap_penalty: 1.0
1 MTPVRRS-LS  9          Score: (32.0-1.0) = 31.0
```

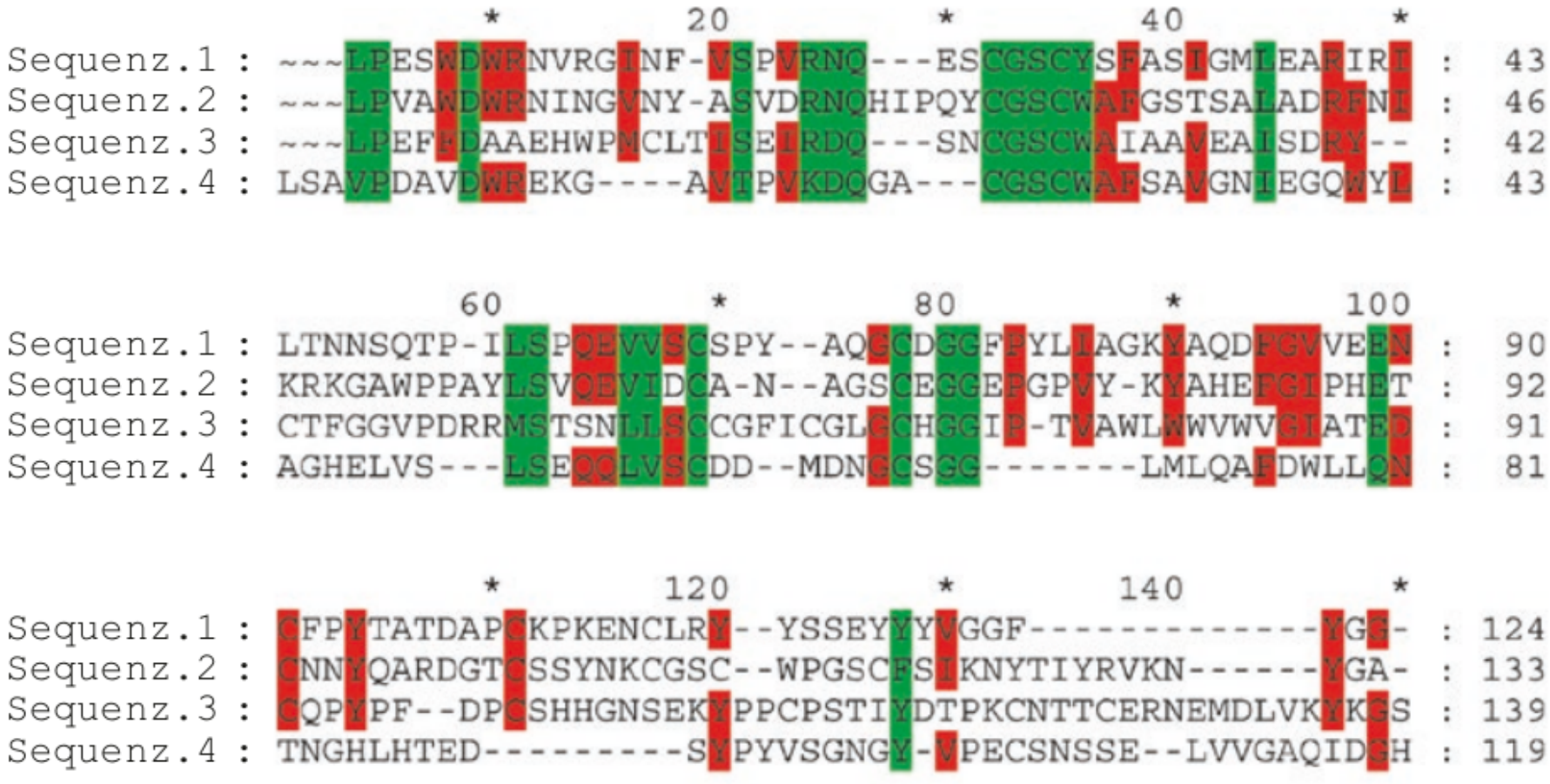
Local alignment

- **Local alignment** is studied when interest is focused only on the most similar stretches within two sequences
- This way, it is possible to identify protein domains and motifs (ATP binding sites, DNA binding domains)
- The local alignment is calculated in the same way as a global alignment using a substitution matrix and the introduction and extension of gaps
- At variance with the global alignment, there is no negative score (replaced by zero)
- The path through the matrix does not move from the lower right to the upper left, but starts and ends at arbitrary places

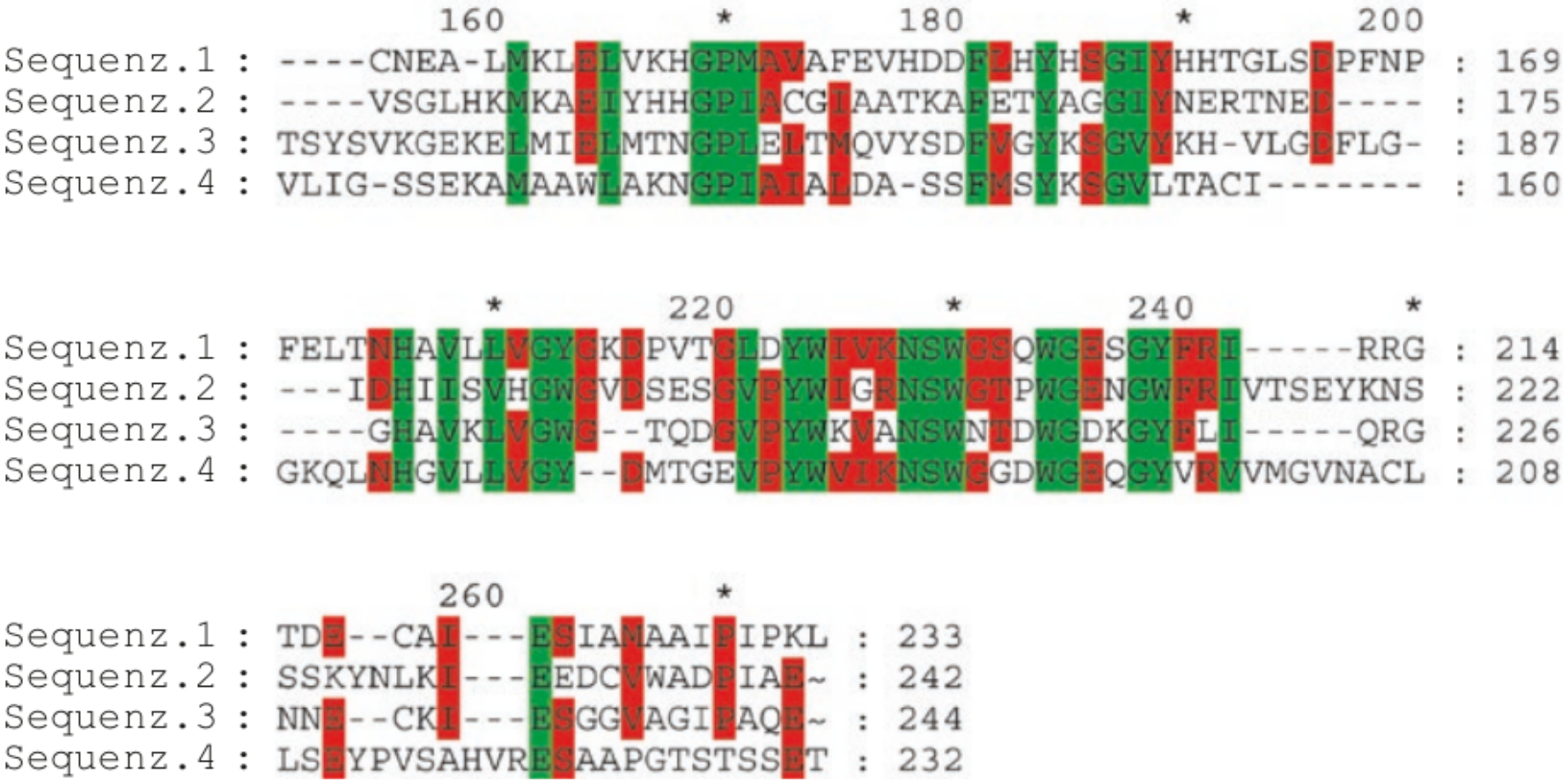
Local alignment

```
XLRHODOP        2     gtagaacagcttcagttgggatcacaggcttctagggatcctttg 46
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1182  gtagaacagcttcagttgggatcacaggcttctagggatcctttg 1226

XLRHODOP        47    ggcaaaaaagaaacacagaaggcattctttctatacaagaaagga 91
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1227  ggcaaaaaagaaacacagaaggcattctttctatacaagaaagga 1271

XLRHODOP        92    ctttatagagctgctaccatgaacggaacagaaggtccaaatttt 136
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1272  ctttatagagctgctaccatgaacggaacagaaggtccaaatttt 1316

XLRHODOP        137   tatgtccccatgtccaacaaaactggggtggtacgaagcccattc 181
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1317  tatgtccccatgtccaacaaaactggggtggtacgaagcccattc 1361

XLRHODOP        182   gattaccctcagtattacttagcagagccatggcaatattcagca 226
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1362  gattaccctcagtattacttagcagagccatggcaatattcagca 1406

XLRHODOP        227   ctggctgcttacatgttcctgctcatcctgcttgggttaccaatc 271
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1407  ctggctgcttacatgttcctgctcatcctgcttgggttaccaatc 1451

XLRHODOP        272   aacttcatgaccttgtttgttaccatccagcacaagaaactcaga 316
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1452  aacttcatgaccttgtttgttaccatccagcacaagaaactcaga 1496

XLRHODOP        317   acacccctaaactacatcctgctgaacctggtatttgccaatcac 361
                      |||||||||||||||||||||||||||||||||||||||||||||
XL23808         1497  acacccctaaactacatcctgctgaacctggtatttgccaatcac 1541
```

Multiple alignment



Multiple alignment



Section 7

Biopython

Introduction

- Biopython is a collection of libraries intended for Python programming in bioinformatics
- The central entity in bioinformatics is the **sequence**
- Biopython provides an object, called `Seq` which allows to consider a string as a biological sequence

```
dna2 = Seq('AGTACACTGGT')
```

```
dna2
```

```
Seq('AGTACACTGGT')
```

```
dna2.alphabet
```

```
Alphabet()
```

Introduction: alphabet

- The alphabet object makes the `Seq` object more than just a thing
- `Bio.Alphabet.IUPAC` provides basic definitions for proteins, DNA and RNA

Introduction: alphabet

Proteins	IUPAC.protein
Proteins	IUPAC.extended_protein
Proteins	IUPAC.ExtendedIUPACProtein
Proteins	IUPAC.protein
Proteins	IUPAC.IUPACProtein()
RNA	IUPAC.IUPACAmbiguousRNA(
RNA	IUPAC.IUPACUnambiguousRNA()
RNA	IUPAC.ambiguous_rna
RNA	IUPAC.unambiguous_rna
DNA	IUPAC.ExtendedIUPACDNA()
DNA	IUPAC.IUPACData
DNA	IUPAC.ambiguous_dna
DNA	IUPAC.unambiguous_dna
DNA	IUPAC.IUPACAmbiguousDNA()
DNA	IUPAC.IUPACUnambiguousDNA()
DNA	IUPAC.extended_dna

Introduction: alphabet

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
```

```
s1 = 'GATCGATGGGCCTATATAGGATCGAAAATCGC'
```

```
dnaseq = Seq(s1, IUPAC.unambiguous_dna)
```

```
dnaseq.alphabet
```

```
IUPACUnambiguousDNA()
```

Introduction: alphabet

```
s2 = 'GAUCGAUGGGCCUAUAUAGGAUCGAAAAUCGC'
```

```
rnaseq = Seq(s2, IUPAC.unambiguous_rna)
```

```
rnaseq.alphabet
```

```
IUPACUnambiguousRNA()
```

```
protseq = Seq(s1,IUPAC.protein)
```

```
protseq.alphabet
```

```
IUPACProtein()
```

Introduction: alphabet

`Seq` objects can be used as if they were normal Python strings

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
```

```
s1 = 'GATCGAT'
```

```
dnaseq = Seq(s1, IUPAC.unambiguous_dna)
```

```
for index, letter in enumerate(dnaseq):
    print(index,"\t",letter)
```

```
0        G
1        A
2        T
3        C
4        G
5        A
6        T
```

Complement of a strand

For nucleotides sequences the **complement**, or **reverse complement** can be obtained using built-in `Seq` functions:

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
```

```
dna1 = 'GATCGATGGGCCTATATAGGATCGAAAATCGC'
```

```
dnaseq = Seq(dna1, IUPAC.unambiguous_dna)
```

```
dnaseq.complement()
```

```
Seq('CTAGCTACCCGGATATATCCTAGCTTTTAGCG', IUPACUnambiguousDNA())
```

```
template_dna = dnaseq.reverse_complement()
```

```
template_dna
```

```
Seq('GCGATTTTCGATCCTATATAGGCCCATCGATC', IUPACUnambiguousDNA())
```

Transcription

DNA coding strand (aka Crick strand, strand +1)

```
5’   ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG   3’
     ||||||||||||||||||||||||||||||||||||||||
3’   TACCGGTAACATTACCCGGCGACTTTCCCACGGGCTATC   5’
```

DNA template strand (aka Watson strand, strand −1)

Transcription

```
5’   AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG   3’
```

Single stranded messenger RNA

Transcription

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
```

```
coddna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG", IUPAC.unambiguous
```

```
coddna
```

```
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
```

```
temdna = coddna.reverse_complement()
```

```
temdna
```

```
Seq('CTATCGGGCACCCTTTCAGCGGCCCATTACAATGGCCAT', IUPACUnambiguousDNA())
```

```
mesrna = coddna.transcribe()
```

```
mesrna
```

```
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
```

```
backcoddna = mesrna.back_transcribe()
```

```
coddna == backcoddna
```

```
True
```

Back transcription

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
```

```
mesrna = Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPAC.unambiguous
```

```
mesrna
```

```
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
```

```
mesrna.back_transcribe()
```

```
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
```

Translation

```
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC
```

```
mesrna = Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPAC.unambiguous
```

```
mesrna
```

```
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG', IUPACUnambiguousRNA())
```

```
mesrna.back_transcribe()
```

```
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG', IUPACUnambiguousDNA())
```

```
mesrna.translate()
```

```
Seq('MAIVMGR*KGAR*', HasStopCodon(IUPACProtein(), '*'))
```

Sequence Input/Output: reading

The `Bio.SeqIO` module aims at providing a simple interface for working with assorted sequence file formats in a uniform way

- To read sequences as SeqRecord, the function `Bio.SeqIO.parse()` can be used
- The first argument of this function is a filename
- The second argument is lower case string specifying the alphabet to be used

Sequence Input/Output: reading

```
from Bio import SeqIO
```

```
for seqrec in SeqIO.parse('ls_orchid.fasta', 'fasta'):
    print(seqrec.id)
    print(repr(seqrec.seq))
    print(len(seqrec))
```

```
...
```

```
for seq_record in SeqIO.parse("ls_orchid.gbk", "genbank"):
    print(seq_record.id)
    print(repr(seq_record.seq))
    print(len(seq_record))
```

Sequence Input/Output: reading (alternative way)

```
record = list(SeqIO.parse('ls_orchid.fasta', 'fasta'))
```

```
print("Found ", len(record), "records")
```

```
Found  94 records
```

```
print(record[0].seq)
```

```
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGTGAA
TCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTTGGGCCGC
CTCGGGAGCGTCCATGGCGGGTTTGAACCTCTAGCCCGGCGCAGTTTGGGCGCCAAGCCATATGAAAGCATC
ACCGGCGAATGGCATTGTCTTCCCCAAAACCCGGAGCGGCGGCGTGCTGTCGCGTGCCCAATGAATTTTGAT
GACTCTCGCAAACGGGAATCTTGGCTCTTTGCATCGGATGGAAGGACGCAGCGAAATGCGATAAGTGGTGTG
AATTGCAAGATCCCGTGAACCATCGAGTCTTTTGAACGCAAGTTGCGCCCGAGGCCATCAGGCTAAGGGCAC
GCCTGCTTGGGCGTCGCGCTTCGTCTCTCTCCTGCCAATGCTTGCCCGGCATACAGCCAGGCCGGCGTGGTG
CGGATGTGAAAGATTGGCCCCTTGTGCCTAGGTGCGGCGGGTCCAAGAGCTGGTGTTTTGATGGCCCGGAAC
CCGGCAAGAGGTGGACGGATGCTGGCAGCAGCTGCCGTGCGAATCCCCCATGTTGTCGTGCTTGTCGGACAG
GCAGGAGAACCCTTCCGAACCCCAATGGAGGGCGGTTGACCGCCATTCGGATGTGACCCCAGGTCAGGCGGG
GGCACCCGCTGAGTTTACGC
```

Sequence Input/Output: writing

- To write sequences as SeqRecord, the function `Bio.SeqIO.write()` can be used
- The first argument of this function is a `SeqRecord` (the information we want to write)
- The second argument is a filename
- The third argument is a sequence format

Sequence Input/Output: writing

```
from Bio import SeqIO
from Bio.SeqRecord import SeqRecord
```

```
rec1 = list(SeqIO.parse('Q93XP8.fasta', 'fasta'))
rec2 = list(SeqIO.parse('Q94IT9.fasta', 'fasta'))
rec3 = list(SeqIO.parse('Q94JN8.fasta', 'fasta'))
```

```
print(rec1[0].id, "\n",rec2[0].id,"\n", rec3[0].id)
```

```
tr|Q93XP8|Q93XP8_TOBAC
 tr|Q94IT9|Q94IT9_FRAVE
 tr|Q94JN8|Q94JN8_CUCSA
```

```
rec1seq = SeqRecord(rec1[0].seq, rec1[0].id, rec1[0].description)
rec2seq = SeqRecord(rec2[0].seq, rec2[0].id, rec3[0].description)
rec3seq = SeqRecord(rec3[0].seq, rec3[0].id, rec3[0].description)
```

```
new_rec = [rec1seq, rec2seq, rec3seq]
```

```
SeqIO.write(new_rec,'chalcone.faa', 'fasta')
```

Sequence alignment

- Biopython provides a set of functions for local and global pairwise alignments, `Bio.pairwise2`
- In the following slides, the alignment of two hemoglobin sequences (HBA_HUMAN and HBB_HUMAN) are stored in `alpha.fasta` and `beta.fasta`
- The alignment function is `align.globalxx`: the last two letters of the function name (`xx`) are used for decoding the scores and the penalties for matches/mismatches and gaps

Sequence alignment

- In this case, the first letter decodes the match score, namely `x` means that a match counts 1 while mismatches have no costs
- The second letter decodes for the cost of gaps: `x` means no gap costs at all
- `globalxx` means that only matches between both sequences are counted

Sequence alignment

```
from Bio import pairwise2
from Bio import SeqIO
```

```
s1 = SeqIO.read('P68871.fasta', 'fasta')
s2 = SeqIO.read('P69905.fasta', 'fasta')
```

```
print(s1.seq)
print(s2.seq)
print(len(s1.seq))
print(len(s2.seq))
```

```
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLGAF
SDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANALAH
KYH
MVLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVA
HVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKYR
147
142
```

Sequence alignment

```
alignments = pairwise2.align.globalxx(s1.seq, s2.seq)
```

```
len(alignments)
```

```
135
```

```
alignments[0]
```

```
('MVHLTPEEKS-AV--T---AL-WGKVNVDEVG---GE--A--LG-RL--LVVY--PWTQRF----FES--
FGDLSTPDAVM-GNPK---VKA-HGKKVLGAFSDG-L--A--HL-D---N-LKGTF-ATLSE-LHCD--KLH
-VDPE-NFRL-LGNVLV--C--V-LAH-HFGK---EFTPPVQA--AYQ---KVV--AG-VANA---LAH--K
YH-',
 'MV-L-----SPA-DKTNVKA-AWGK-----VGAHAGEYGAEAL-ER-MFL---SFP-T---TKTYF--PH
F-DLS------HG---SAQVK-GHGKKV--A--D-ALTNAVAH-VDDMPNAL----SA-LS-DLH--AHKL-
RVDP-VNF--KL---L-SHCLLVTLA-AH---LPAEFT-P--AVHA--SLDK--FLA-SV---STVL--TSK
Y-R',
 72.0,
 0,
 217)
```

Sequence alignment

```
print(alignments[0][0][:40])
print(alignments[0][1][:40])
```

```
MVHLTPEEKS-AV--T---AL-WGKVNVDEVG---GE--A
MV-L-----SPA-DKTNVKA-AWGK-----VGAHAGEYGA
```

```
print(pairwise2.format_alignment(*alignments[0]))
```

```
MVHLTPEEKS-AV--T---AL-WGKVNVDEVG---GE--A--LG-RL--LVVY--PWTQRF----FES--FG
DLSTPDAVM-GNPK---VKA-HGKKVLGAFSDG-L--A--HL-D---N-LKGTF-ATLSE-LHCD--KLH-V
DPE-NFRL-LGNVLV--C--V-LAH-HFGK---EFTPPVQA--AYQ---KVV--AG-VANA---LAH--KYH
-
||  |      | |    |    |   |||      ||    ||   |   |   |    |      | |         |     |
|||         |       ||   |||||   |   |   |   |   |    | |      | ||   ||     ||   |
||   ||    |    |    |   | ||   |       |||  |   |   |      |     |   |        |     ||

MV-L-----SPA-DKTNVKA-AWGK-----VGAHAGEYGAEAL-ER-MFL---SFP-T---TKTYF--PHF-
DLS------HG---SAQVK-GHGKKV--A--D-ALTNAVAH-VDDMPNAL----SA-LS-DLH--AHKL-RV
DP-VNF--KL---L-SHCLLVTLA-AH---LPAEFT-P--AVHA--SLDK--FLA-SV---STVL--TSKY-
R
  Score=72
```

Sequence alignment (with penalty)

```
from Bio import pairwise2
from Bio import SeqIO
from Bio.SubsMat.MatrixInfo import blosum62
```

```
s1 = SeqIO.read('P68871.fasta', 'fasta')
s2 = SeqIO.read('P69905.fasta', 'fasta')
```

```
alignments = pairwise2.align.globalds(s1.seq, s2.seq, blosum62, -10, -0.5
```

```
print(pairwise2.format_alignment(*alignments[0]))
```

```
MVHLTPEEKSAVTALWGKV--NVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVANAL
AHKYH
|| |.|..|..|.|.|||| ...|.|.|||.|.....|.|...|..| |||     .|...||.|||||..
|.....||.|........||.||..||.|||.||.||...|...||.|...||||.|.|...|..|.|...
|..||.
MV-LSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHF-DLS-----HGSAQVKGHGKKVAD
ALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVL
TSKYR
  Score=292.5
```

Pairwise sequence alignment

```
from Bio import pairwise2
from Bio import SeqIO
from Bio import Align
```

```
s1 = SeqIO.read('P68871.fasta', 'fasta')
s2 = SeqIO.read('P69905.fasta', 'fasta')
```

```
aligner = Align.PairwiseAligner()
```

```
score = aligner.score(s1.seq, s2.seq)
```

```
score
```

```
72.0
```

Pairwise sequence alignment

```
from Bio import pairwise2
from Bio import SeqIO
from Bio import Align
```

```
aligner = Align.PairwiseAligner()
```

```
xalign = aligner.align('GAACTCG', 'GAT')
```

```
for x in xalign:
    print(x)
```

```
GAACTCG
||--|--
GA--T--

GAACTCG
|-|-|--
G-A-T--
```