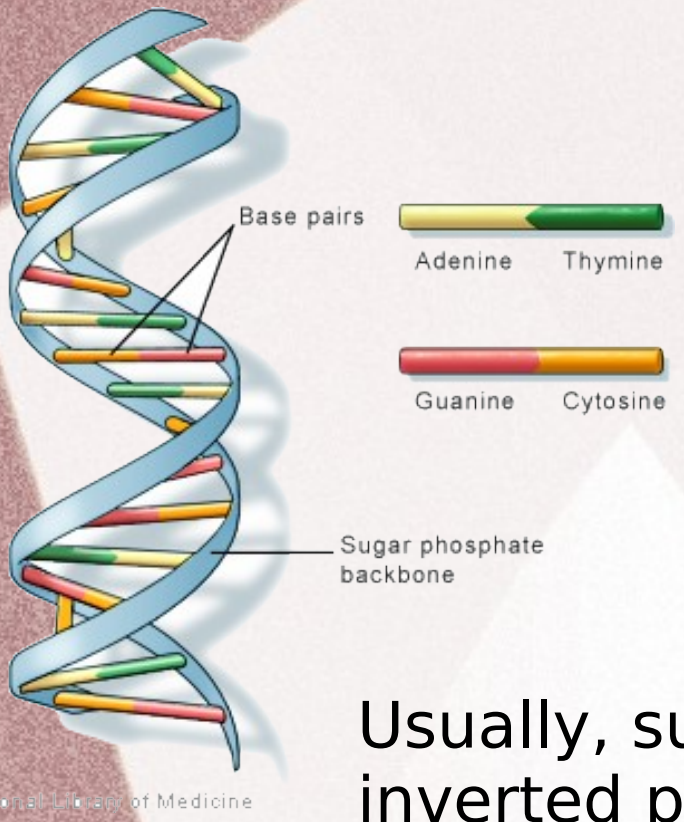
Lecture 5

- More examples of work with strings:
  Exercise 2.37 DNA mutation (pages 86-87)
- More about lists:
  List Comprehension (section 4.5)
- Calling Functions (section 2.6)
- Python Modules (section 2.7)
- Expressions (section 2.8)

Exercise 2.37 DNA mutation



Recall that DNA can be modeled as a string of characters using alphabet: A,C, G, and T

One of the forms of DNA mutation: is when a substring of the DNA is reversed during replication process.

Usually, such a reversal occurs between inverted pairs:
for example, it is possible that between TGAA and AAGT a slice of DNA is inverted and re-attached

Exercise 2.37 DNA mutation

Design a program that works as follows:

- asks the user for an original DNA string, and a particular pattern that is inverted, then

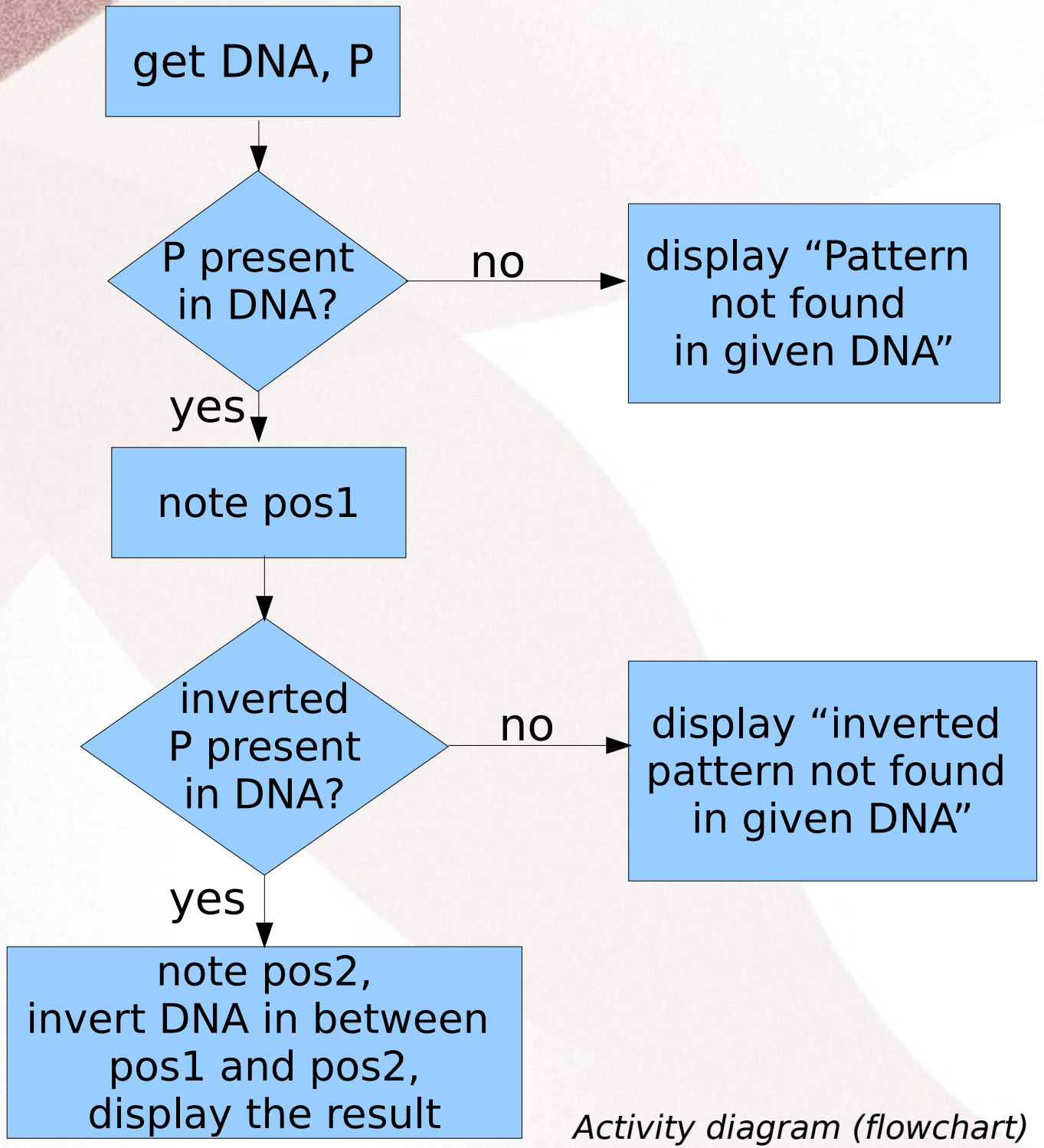
- locates the leftmost occurrence of that pattern, and the next subsequent occurrence of the inverted pattern,

- the output should be the mutated DNA, i.e. segment between the inverted pair is reversed

Example of a session:

Enter a DNA sequence: CGATTGAACATTAAGTCCAATT
Enter the pattern: TGAA
Mutated DNA sequence: CGATTGAATTACAAGTCCAATT

an initial design of the program:



Activity diagram (flowchart)

see the implementation in mutatedDNA.py

4.5 List Comprehension

Assume that we have a list of people invited for a party:

```
guests = ['Tom','Mary','Lily','James']
```

Compare the following two pieces of code:

```
guests_lc= []
for person in guests:
    guests_lc.append(person.lower())
```

and

```
guests_lc = [person.lower() for person in guests]
```

Syntax:
result = [*expression* **for** *identifier* **in** *sequence*]

4.5 List Comprehension

Syntax:
result = [*expression* **for** *identifier* **in** *sequence*]

More **General Syntax**:
result = [*expression* **for** *identifier* **in** *sequence* **if** *condition*]

it is the same as:

```
result = []
for identifier in sequence:
    if condition:
        result.append(expression)
```

4.5 List Comprehension

Example: exercise 4.38 on page 156

Given a list **orig**, possibly containing duplicate values, show how to use *list comprehension* to produce a new list **uniq** that has all values from the original but with duplicated omitted.

4.5 List Comprehension

Example: exercise 4.38 on page 156

Given a list **orig**, possibly containing duplicate values, show how to use *list comprehension* to produce a new list **uniq** that has all values from the original but with duplicated omitted.

```
uniq = [item for item in orig if item not in uniq]
```

4.5 List Comprehension

Example: exercise 4.38 on page 156

Given a list **orig**, possibly containing duplicate values, show how to use *list comprehension* to produce a new list **uniq** that has all values from the original but with duplicated omitted.

```
uniq = [item for item in orig if item not in uniq]
```

- is not working

4.5 List Comprehension

Example: exercise 4.38 on page 156

Given a list **orig**, possibly containing duplicate values, show how to use *list comprehension* to produce a new list **uniq** that has all values from the original but with duplicated omitted.

```
uniq = [item for item in orig if item not in
uniq]
```

- is not working

How about this one? :

```
l=len(orig)
Uniq=[ orig[i] for i in range(l) if
                    orig[i] not in orig[i+1:]]
```

4.5 List Comprehension

See programs exercise_4-38.py,
exercise_4-38-mod1.py, and
exercise_4-38-mod2.py

2.6 Calling Functions

Usually when we *call* (*invoke*) an object's method we use the following syntax:

```
object.method(parameters)
```

Also we saw functions that are called outside the context of a particular object or class, for example:

```
def main():
    hello("Ingrid")

def hello(person):
    print("Hello, ",person+".","How are you?")

main()
```

Note, that we didn't call `person.hello()`

- we call such functions *pure functions*, to distinguish them from member function defined as a part of class.

2.6 Calling Functions

Examples of *pure functions*:

```
ord
chr
len
range
max
min
sum
round
pow
abs
```

-look for descriptions of these functions on page 62

and there are many more built-in functions

2.7 Python Modules

In addition to the bunch of built-in functions (available from the very start of the Interpreter), there are lots of other useful tools, that were written by developers of Python and other Python users.

They are usually placed into libraries, called modules, that can be individually loaded as needed.

Recall `math` and `random` libraries.

Whenever we need functions/methods from such modules, we *import* them.

2.7 Python Modules

Three ways of importing:

1. import module as a whole: `import math`
- all the structures (functions, classes, constants) are imported from this module, but

- in order to use a function from this class one needs to do the following: `math.sqrt(2)`

2. handpick the things we need from the module:

```
from math import sqrt, pi
```

Then, they are *directly* available and one can call `sqrt(2)`

3. import *everything* from the module:

```
from math import *
```

Everything from this module can be used directly, without giving a qualified name, as in case 1.

```
sqrt(2)
```

2.7 Python Modules

See page 63 for the short list of modules

2.8 Expressions

When there are more than two operations in an expression, some determination must be made as to which action is performed first.

We say: an operation performed first is given precedence over the others.

Precedence of operations in Python:

highest precedence

Operator	Description
(expr), [expr], {key:datum...}, `expressions...`	*Binding or tuple display, list display, dictionary display, string conversion*
x[index], x[index:index], x(arguments...), x.attribute	*Subscription, slicing, call, attribute reference*
**	*Exponentiation (right-associative)*
+x, -x, ~x	*Positive, negative, bitwise NOT*
*, /, //, %	*Multiplication, division, quotient, remainder*
+, -	*Addition and subtraction*
<<, >>	*Shifts*
&	*Bitwise AND*
^	*Bitwise XOR*
|	*Bitwise OR*
in, not in, is, is not, <, <=, >, >=, <>, !=, ==	*Comparisons, including membership tests and identity tests,*
not x	*Boolean NOT*
and	*Boolean AND*
or	*Boolean OR*
lambda	*Lambda expression*

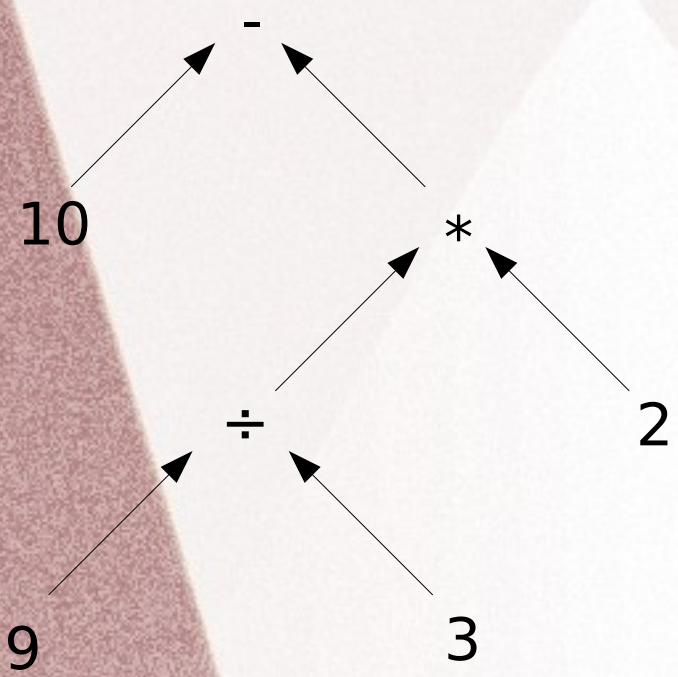
lowest precedence

2.8 Expressions

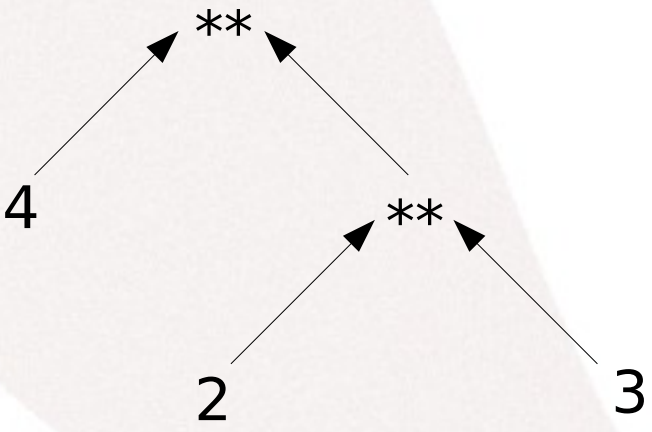
For a given expression we can build an evaluation tree portraying the evaluation order graphically.

Examples:

10-9÷3*2



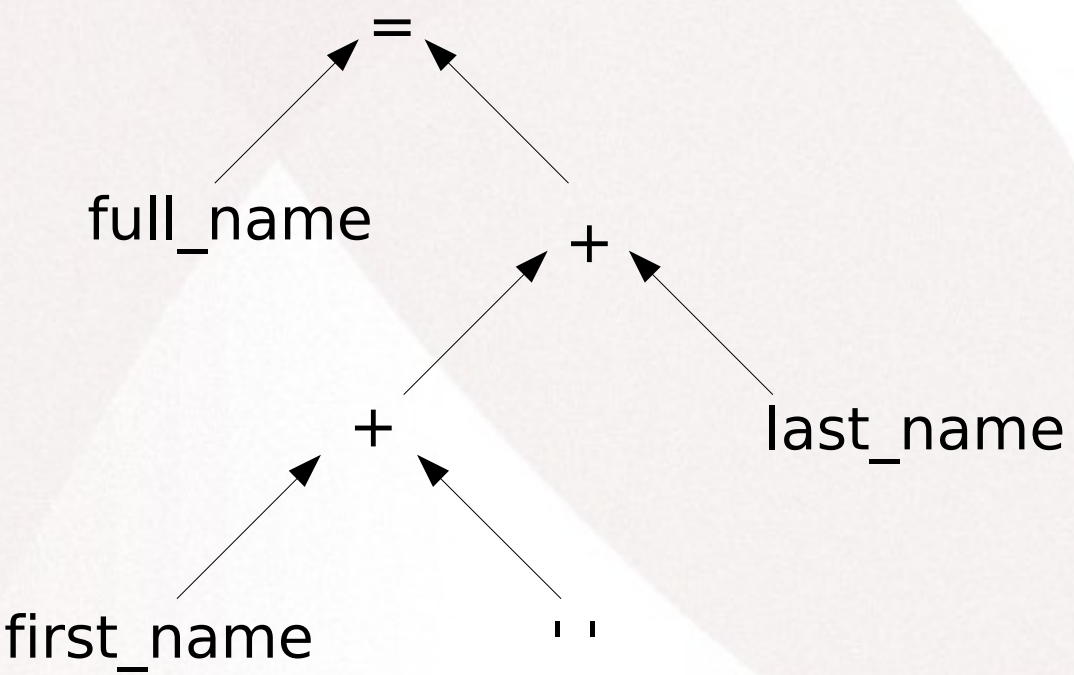
4**2**3 = 4 ** (2**3)



2.8 Expressions

one more example:

full_name = first name + ' ' + last_name



! The **assignment operator** has the **lowest precedence**

Boolean Expressions and the bool Class

Called in the name of George Boole who pioneered the study of many logical properties.

T	True	1		Holds
F	False	0	$\perp$ (bottom)	Doesn't hold

Usually used to represent some logical condition that is assumed to be either true or false.

Few of the logical operators:

Operator		Python Syntax	Description
$\neg x$	$\overline{x}$	not x	negation of x
$A \wedge B$		A and B	conjunction of A and B
$A \vee B$		A or B	disjunction of A and B

Boolean Expressions and the bool Class

Truth table for the three operators:

x	y	not x	x and y	x or y
T	T	F	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	F

x and y is **True** only when both x, and y are True

x or y is **False** only when both x, and y are False

Boolean Expressions and the *bool* Class

Examples:

written in math

$60 \leq \text{Temp} \leq 80$

in Python:

```
60 <= Temp and Temp <= 80   or
Temp >= 60 and Temp <= 80   or
60 <= Temp <= 80
```

Length of groceries list is not more than 10 and milk is in the list

```
len(groceries) <= 10 and
                  'milk' in groceries
```

In-class work

- Using *list comprehension* create an array(list) of *n* elements of the following form: $[1, \frac{1}{3}, \frac{1}{5}, \frac{1}{7}, \frac{1}{9}, \frac{1}{11}, \frac{1}{13}, \ldots, n^{th}\ term]$ $n \geq 1$
- Predict what list will be generated in the following block of code:

```
a = 10
myList = [pow(-1,i)*i/pow(2,i%10) for i in range(a)]
```

write the list of numbers in fraction form.

- Draw an evaluation tree for the following expressions:

a) $17 - \frac{4^2}{\sqrt[3]{81}}$ b) “High”.lower()+” 503”.remove('0')

- Draw the truth table for the following proposition: $\neg(p \vee \neg q) \wedge r$

Homework Assignment

1. Exercise 4.35 on page 156

2. Exercise 2.27 on page 85

3. Draw an evaluation tree for the following statements:

   (a) $23 + 10 (2^2 * 3 - \sqrt{64})$
   (b) exercise 2.29 on page 85

4. Do exercise 2.36 on page 86