V:
Università
degli Studi
della Campania
Luigi Vanvitelli

Exercise

Ing. Lelio Campanile

Factorial

the factorial of a positive integer, denoted by n!, is the product of all positive integers less than or equal to n:

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \ldots \times 3 \times 2 \times 1.$$

Recursion

method of solving a problem where the solution depends on solutions to smaller instances of the same problem

In python you could implement it witha function that calls itself.

Recursive Factorial

```python
def factorial_rec(n):
    if n == 0:
        return 1
    else:
        return n * factorial_rec(n-1)
```

Iterative Factorial

```python
def factorial(n):
    for i in range((n-1), 0, -1):
        n = n * i
    return n
```

Fibonacci Numbers

the Fibonacci numbers, commonly denoted Fn form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from 0 and 1:

$F_0 = 0$, $F_1 = 1$ and
$F_n = F_{n-1} + F_{n-2}$

Recursive Fibonacci

```python
def fibonacci_rec (n):
    if n == 0 or n == 1:
        return 1
    else:
        return fibonacci_rec(n-1) + fibonacci_rec(n-2)
```

Iterative Fibonacci

```python
def fibonacci(n):
    if n == 0:
        return n
    last = 0
    next = 1
    for _ in range(1,n+1):
        previous_last = last
        last = next
        next = previous_last + last
    return next
```

Pigreco - Leibniz Formula

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + \frac{(-1)^n}{2n+1} + \cdots = \frac{\pi}{4}.$$

Simplified version to calculate the convergence of π

$$\pi = 4/1 - 4/3 + 4/5 - 4/7 + 4/11\dots$$

```python
def calculate_pi(n_terms):
    numerator= 4.0
    denominator = 1.0
    operation = 1.0
    pi = 0.0
    for _ in range(n_terms):
        pi += operation * (numerator/denominator)
        denominator += 2.0
        operation *= -1.0
    return pi
```

Searching

Linear Search

```python
def linear_contains(iterable, key):
    for item in iterable:
        if item == key:
            return True
    return False
```

Binary search

only on sorted data

```python
def binary_contains(sorted_iterable, key):
    low = 0
    high = len(sorted_iterable) - 1
    while low <= high:
        mid = (low + high) // 2
        if sorted_iterable[mid] < key:
            low = mid + 1
        elif sorted_iterable[mid] > key:
            low = mid - 1
        else:
            return True
    return False
```

Contacts

```python
if __name__=="__main__":

    print('Contacts')
    print('------------------------------')
    contacts={}
    your_choice= 0
    while your_choice != 5:
        menu()
        your_choice=int(input('Select your option: '))
        print('------------------------------')
        if your_choice==1:
            add_contact(contacts)
        elif your_choice==2:
            remove_contact(contacts)
        elif your_choice==3:
            list_contacts(contacts)
        elif your_choice==4:
            search_contact(contacts)
```

```python
def menu():
    print('1 - add a contact')
    print('2 - remove a contact')
    print('3 - list all contacts')
    print('4 - search a contact')
    print('5 - exit')
```

```python
def add_contact(dict):
    contact=input('Insert name: ')
    mobile_number=input('Mobile number: ')
    dict[contact]=mobile_number
    print('Contact %s with %s number added\n' % (contact,mobile_number))
```

```python
def remove_contact(dict):
    contact = input('Contact to remove: ')
    if contact in dict.keys():
        del dict[contact]
        print('Contact removed')
    else:
        print('Contact %s not found' % (contact))
```

```python
def list_contacts(dict):
    if len(dict)==0:
        print("Nothing here")
    for x in dict.keys():
        print("Contats: %s \tNumber: %s" % (x, dict[x]))
```

```python
def search_contact(dict):
    contact = input('Insert contact to search: ')
    for name in dict.keys():
        if name == contact:
            print('Contact %s with %s number\n' % (name,dict[name]))
    print("Nothing here")
```

DNA

Deoxyribonucleic acid

Genes are commonly represented in computer software as a sequence of the characters cytosine [C], guanine [G], adenine [A] or thymine [T]. Each letter represents a *nucleotide*, and the combination of three nucleotides is called a *codon*.
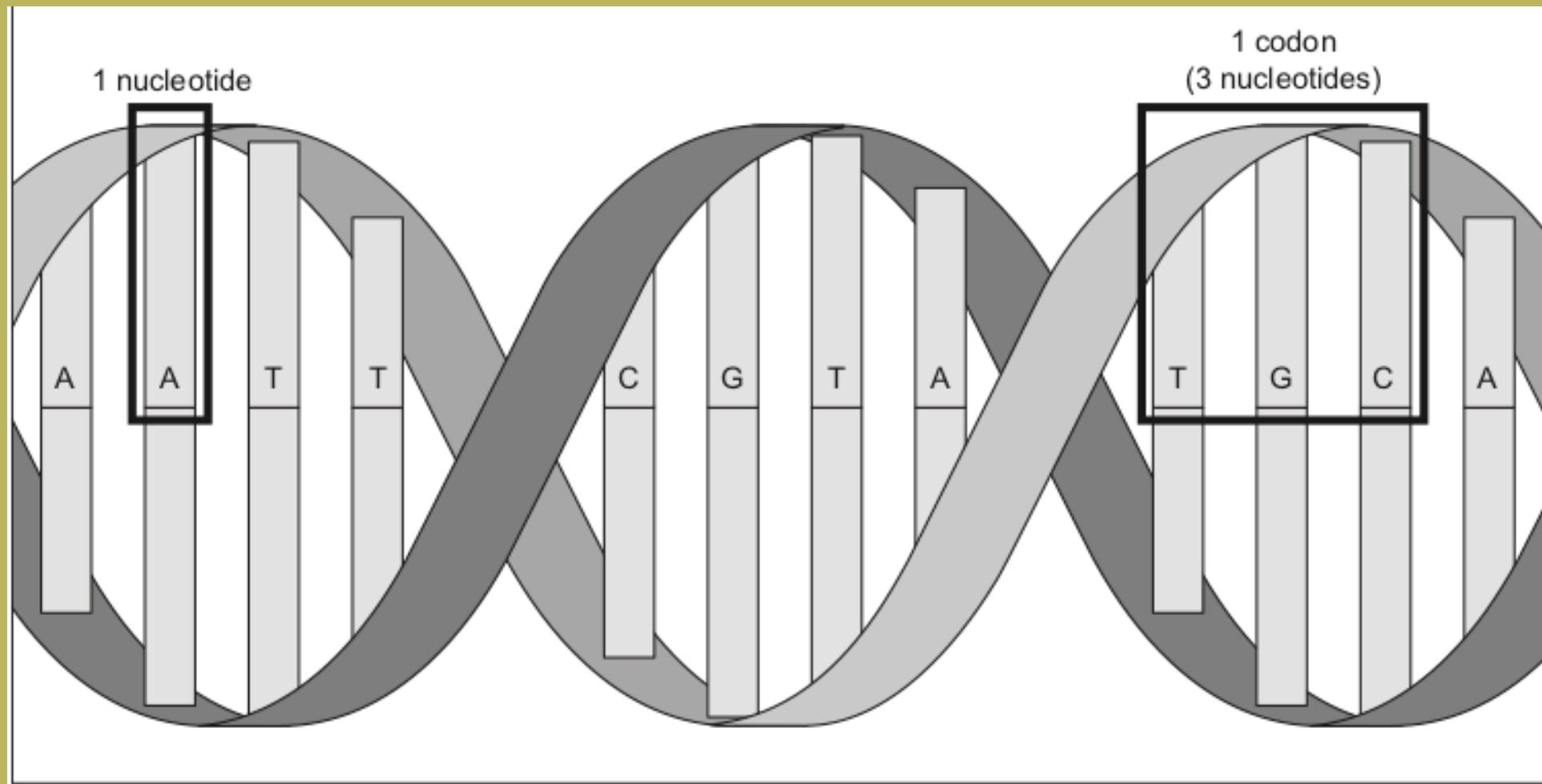
```python
Nucleotide = {'A': 'A', 'C':'C', 'G':'G', 'T':'T'}
gene_str = "ACGTGGCTCTCTAACGTACGTACGTACGGGGGTTTATATATACCCTAGGACTCCCTTT"


def string_to_gene(s):
    gene = []
    for i in range(0, len(s), 3):
        if (i + 2) >= len(s):
            return gene
        codon = (Nucleotide[s[i]], Nucleotide[s[i+1]], Nucleotide[s[i+2]])
        gene.append(codon)
    return gene
```

```python
def linear_contains (gene, key_codon):
    for codon in gene:
        if codon == key_codon: # only for educational purpose
            #else return (key_codon in gene) should be enough
            return True
    return False
```

binary search works only on sorted gene!

```python
def binary_contains(gene, key_codon):
    low = 0
    high = len(gene) - 1
    while low <= high:
        mid = (low + high) // 2
        if gene[mid] < key_codon:
            low = mid + 1
        elif gene[mid] > key_codon:
            high = mid - 1
        else:
            return True
    return False
```

```python
if __name__ == '__main__':
    gene_str = "ACGTGGCTCTCTAACGTACGTACGTACGGGGGTTTATATATACCCTAGGACTCCCTTT"
    my_gene = string_to_gene(gene_str)
    print(my_gene)
    acg = (Nucleotide['A'], Nucleotide['C'], Nucleotide['G'])
    gat = (Nucleotide['G'], Nucleotide['A'], Nucleotide['T'])

    print(linear_contains(my_gene, acg))  # True
    print(linear_contains(my_gene, gat))  # False

    sorted_gene = sorted(my_gene)
    print(binary_contains(sorted_gene, acg)) # True
    print(binary_contains(sorted_gene, gat)) # False
```