**Python Advanced Notebook**

**Helpful Links:**

* [Big O Notations and Time Complexity](https://flexiple.com/algorithms/big-o-notation-cheat-sheet)

**Theoretical Tests:**

* [Lists as Stacks and Queues](https://forms.gle/XP8QW5K59yMGQE7N8)
* [Tuples and Sets](https://forms.gle/AW52AwMsgNbvQfMf7)
* [Multidimensional Lists](https://forms.gle/h5rchwrtc4jaanzE6)
* [Functions](https://forms.gle/ZEXu7H1VAidyxMx19)
* [Error Handling](https://forms.gle/kz2vCqLuZ8MJhZvX6)
* [File Handling](https://forms.gle/kPob4JXS4pcF75J27)
* [Modules](https://forms.gle/xCie5pdDdNKXmSWS8)

**Notebook:**

**01. Lists as Stacks and Queues (Списъци като Стекове и Опашки)**

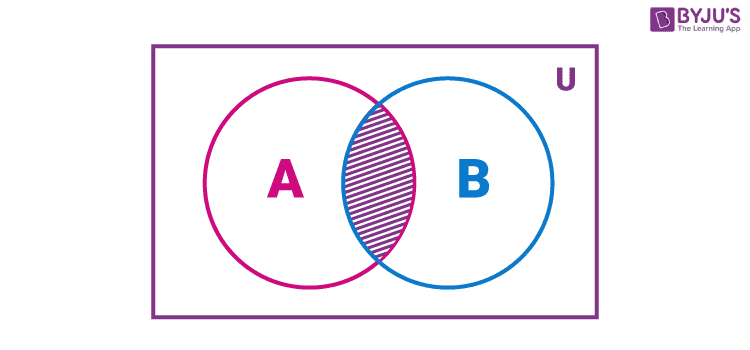
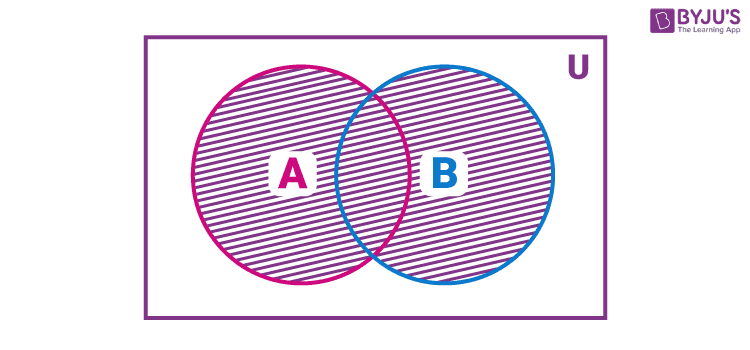
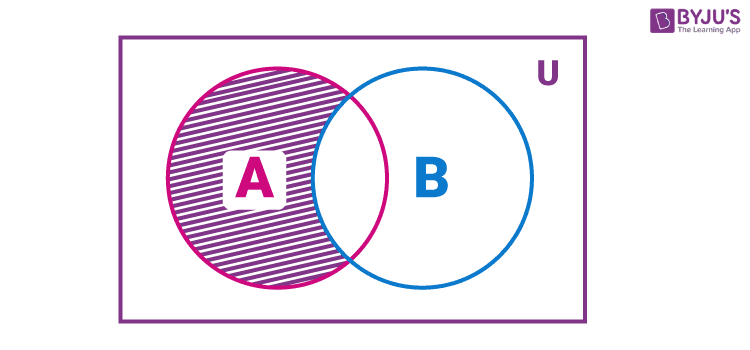
1. Time Complexity
   * Броя операции при определен брой входни данни
   * Използваме го, за да измерваме бързодействието на един алгоритъм
   * Инструменти за измерване:
     + Big O notation - показва **worst case** сложност на алгоритъма
     + Big – Theta(Θ) notation - показва **average case** сложност на алгоритъма
     + Omega notation - показва **best case** сложност на алгоритъма
2. Stack - Стек
   * Линейна структура -> всеки елемент освен първия и последния, има предишен и следващ
   * **LIFO** order -> **L**ast element **I**n **F**irst element **O**ut
   * Типично се реализира с **Linked List**(Свързан списък)
   * В Python можем да използваме списък, за да постигнем подобен ефект, благодарение на методите **append**, **pop**
3. Queue - Опашка
   * Линейна структура
   * Може да бъде реализирана, както с **Linked List**, така и с **Double Linked List**
   * В Python queue е реализиран с **Double Linked List**
   * **FIFO** order -> **F**isrt element **I**n **F**irst element **O**ut

**02. Tuples and Sets (Кортежи и множества)**

1. Tuples (Кортежи)
   * Read-Only collection
   * Immutable (не могат да бъдат променяни)
   * Синтаксис:
   * tuple1 = (1, )

tuple2 = (1, 2, 3)

1. Sets (Множества)

* Неподредена колекция
* Пази само уникални стойности
* Изобразяват се чрез Вен диаграми
* Операции върху множества:
  + intersection (сечение) - A ∩ B - всички общи елементи от А и В  
    [](https://camo.githubusercontent.com/09c3d4e1e1001c49076c0a56dbf40621205f17a53466d320f0928a941016b486/68747470733a2f2f63646e312e62796a75732e636f6d2f77702d636f6e74656e742f75706c6f6164732f323032312f30352f56656e6e2d6469616772616d732d362e706e67)
  + union (обединение) - А U В - всички елементи от А и В  
    [](https://camo.githubusercontent.com/d9eca73c34452e51a65e30d186f56aaa4e8aaf11dfb0dd43c94bd10e04b9e83b/68747470733a2f2f63646e312e62796a75732e636f6d2f77702d636f6e74656e742f75706c6f6164732f323032312f30352f56656e6e2d6469616772616d732d372e706e67)
  + difference (разлика) - А - В - всички различни елементи в А, които не присъстват в В  
    [](https://camo.githubusercontent.com/31dd2917d8c3d2a01fd298b663c4a946e1008db1757e34f5a7eeb6fe29ea171a/68747470733a2f2f63646e312e62796a75732e636f6d2f77702d636f6e74656e742f75706c6f6164732f323032312f30372f646966666572656e63652d6f662d612d616e642d622e706e67)
  + symmetrical difference (симетрична разлика) - A ^ B - всички различни елементи от А, които не присъстват в В и всички елементи от В, които не присъстват в А.
* Синтаксис:
* my\_set\_1 = {1, 2, 3, 4}
* my\_set\_2 = {3, 4, 5, 6}
* print(my\_set\_1.intersection(my\_set\_2)) # {3, 4}
* print(my\_set\_1.union(my\_set\_2)) # {1, 2, 3, 4, 5, 6}
* print(my\_set\_1.difference(my\_set\_2)) # {1, 2}
* print(my\_set\_2.difference(my\_set\_1)) # {5, 6}
* print(my\_set\_1.symmetric\_difference(my\_set\_2)) # {1, 2, 5, 6}

print(my\_set\_2.symmetric\_difference(my\_set\_1)) # {1, 2, 5, 6}

**03. Multidimensional Lists (Многомерни Списъци)**

1. Какво представляват многомерните списъци?
   * Списък от списъци
   * matrix = [
   * [1, 2, 3],
   * [4, 5, 6],
   * [7, 8, 9],

]

1. Как достъпваме даниите в многомерен списък?
2. print(matrix[0][0] # 1
3. print(matrix[1][1] # 5

print(matrix[2][0] # 7

* + Казвайки matrix[0], достъпваме списъка на първа позиция, т.е. [1, 2, 3]
  + Казвайки matrix[0][0], достъпваме елемента на индекс **0** в списъка намиращ се на индекс **0**, във външния списък.

**04. Functions Advanced**

1. Пакетиране на аргументи.
   * Понякога искаме функция да може да приема различен брой аргументи при всяко извикване.
   * Начина, по който постигаме това е чрез пакетиране
2. def find\_average(\*nums): # подадените n на брой параметри ще бъдат пакетирани в списък с променлива на име nums
3. # nums => [10, 11, 9, 7, -12, 56]
4. return sum(nums) / len(nums)

find\_nums(10, 11, 9, 7, -12, 56)

* + Освен стандартни аргумети можем да пакетираме и key-word аргументи
  + За разлика от стандартните те биват запазени в речник

def print\_people\_data(\*\*names): # {ivan: 20, emily: 32, pesho: 16}

for name, age in names.items():

print(f"{name} is {age} years old")

print\_people\_data(ivan=20, emily=32, pesho=16)

* + Ако искаме да подадем определен брой единични аргумети, то можем да го направим, стига да бъдат подадени като първи

def my\_func(param1, param2, \*args, \*\*kwargs):

# param1 => 1

# param2 => hi

# args => [32, 31]

# kwargs => {name: pesho}

pass

my\_func(1, "hi", 32, 31, name="pesho")

1. Разопаковане
   * Освен да пакетираме аргументи, то можем и да ги вадим от този "пакет"
2. numbers = [1, 2, 3]

print(\*numbers) # => \*numbers ще извади всяка стойност и ще я подаде като отделен аргумент => print(1, 2, 3)

1. Анонимни функции
   * lambda(анонимните) функции използваме предимно за кратки изрази
2. lambda a, b: a + b
3. # еквивалетна функция
4. def sum(a, b)

return a + b

1. Вложени функции
   * Вложените функции ни позволяват да достъпваме променливи от scope-a над тях
2. def outer\_func():
3. a = 5
4. def inner\_func():
5. print(a)
6. inner\_func()

outer\_func() # console => 5

1. Recursion
   * Функция, която извика себе си, има дъно и пълни stack-a
   * Кога използваме рекурсия?
     + Когато искаме да използваме stack-a, който идва от машината
   * **Всеки рекурсивен проблем, може да бъде решен итеративно**
2. def fibonacci(n):
3. if n <= 1:
4. return n

return fibonacci(n - 1) + fibonacci(n - 2)

**05. Error Handling**

1. Грешки и Изключения (Errors and Exceptions)
   * Какво е грешка?
     + Проблем или дефект, който се появява в програмата, водейки до неочаквано поведение.
   * Какво е изключение?
     + Събитие, което се случва по време на изпълнение на програмата и нарушава нейното изпълнение
2. Типове грешки:
3. Обработване на грешки
   * С логическите блокове **try**, **except**, ние можем да улавяме грешки
4. def divide\_numbers(x, y):
5. try:
6. result = x / y
7. except ZeroDivisionError: # catches the error, must be included
8. print("Error: You can't divide by zero.")
9. else: # prints only in the case that we don't have an error, optional
10. print(f"The result is {result}")
11. finally: # prints no matter the result from the try-except block, optional

print("Execution of divide\_numbers is complete.")

1. Custom Exceptions
   * Създаваме клас, който задължително наследява **Exception**
2. class MyException(Exception):
3. pass

raise MyException # извикваме грешката

**06. File Handling**

1. Отваряне на файлове
   * Използваме функцията **open()**
   * Тя приема 2 параметъра
     + Абсолютен или относителен път до файл
     + Тип на отваряне на файла
       - **w** - отваря файла за писане, като презаписва предишните данни в него
       - **x** - създава нов файл и го отваря за писане, ако вече има файл, хвърля грешка
       - **a** - отваря файла за писане, но добавя към предишните данни, вместо да ги изтрива
       - **t** - отваря файла в текстове режим, за писане и четене
       - **b** - binary mode
       - **+** - добавя четене или писане към съответния режим. Пример: **rw+**
2. FileNotFoundError
3. try:
4. text\_file = open('text.txt', 'r')
5. print("File found")
6. except FileNotFoundError:

print("File not found")

1. Методи
   * readline - прочита n на брой символа
2. file = open("text.txt") # 'Hello, SoftUni!'

print(file.readline(5)) # 'Hello

* + readlines - прочита всички редове и връща списък

file = open("text.txt")

print(file.readlines())

# ['Every\n', 'Word\n', 'is\n', 'line']

* + close - затваря файла - **опитваме се да не забравяме затварянето на отворени файлове**

file = open('python.txt', 'w')

# Creates or opens the file

file.write("This is the write command.\n")

file.write("It allows us to write in a particular file")

file.close()

1. With statement
   * Отваря файла и когато излезем от блока, го затваря автоматично
2. with open("file.txt", "w") as f:

f.write("Hello World!!!")

1. Изтриване на файл
2. import os
3. os.remove("python.txt")

os.remove("D:\\text.txt")

**07. Modules**

1. Кавко е модул в Python?
   * Python файловете, с които работим наричаме модули
2. Какво наричаме пакет?
   * Пакетите са колекция от модули
   * Те имат **init** файл, в които често се импортват останалите файлове така че да са по-лесно достъпни
3. Видове Модули:
   * Built-In
     + math
     + tkinter
     + os
     + etc.
   * External - [The Python Package Index - PyPI](https://pypi.org/)
4. Запазване на списък с пакети използвани в проекта

pip freeze > requirements.txt

1. Инсталиране на записаните пакети в **requirements.txt**

pip install -r requirements.txt

1. Import modules

from mymodule import myfunc

import mymodule

import mymodule as my