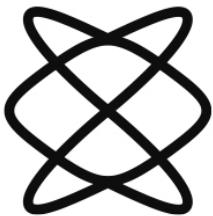


הchg למדעי המחשב (0368)  
פרויקט תוכנה (2161)  
(גרסה ארוכה)

מטרים: ראמי נאסר ורודד שרגן  
תשפ"ג, סמסטר ב' (2023)

מסכם: רועי מעין



The Raymond and  
Beverly Sackler Faculty  
of Exact Sciences  
Tel Aviv University



## פרק 1 – שפת C

3 .....	מבוא
5 .....	שפת C
10 .....	המשך שפת C
15 .....	ממתק בין C ל-Python

## פרק 2 – Python-ב' Data Science

19 .....	NumPy
22 .....	Pandas
26 .....	Machine Learning

## 1 – שפת C

### מבוא

#### עבודה עם Linux

מערכת הפעלה, נעבד אותה על שרת Nova, מתחברים באמצעות ssh. נעבד על ה-shell, לשם כך נלמד פקודות בסיסיות לעבודה עם מערכת הפעלה.

#### מערכת הקבצים של Linux:

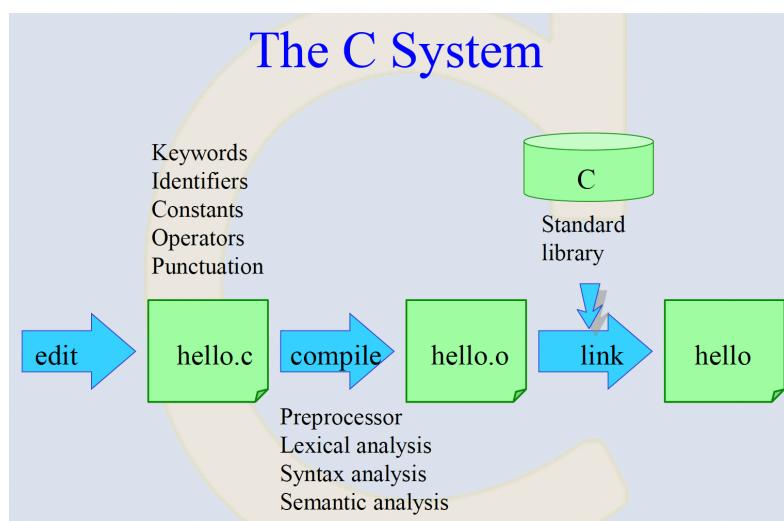
- בהיררכיית הקבצים, הראשון הוא root, ותحتיו נמצאים כל שאר ה-directories.
- פקודות נפוצות: pwd, ls, cd, mkdir, cat, echo, man, mv, cp, grep
- אפשר להעתיק באמצעות scp ממקום למקום מרוחק (באמצעות ssh).
- ניתן לקרוא קבצים באמצעות less או lesspage /, לעבור לתוכאה הבאה עם n, ולצאת עם q.
- ניתן לבצע piping כדי להעביר פלט של תוכנה אחת לקלט של אחרת.

#### סוגיות נוספות:

- נשים לב שכאשר נבצע ctrl+c נשלח SIGINT לתכנית והיא תעצור. כאשר נרצה לעצור תכנית שרצה ברקע, לא יוכל להשתמש ב-c+d ונתממש ב-SIGKILL באמצעות הפקודה kill -9. שימוש lesspage | grep | ps לשם התוכנית שרצה.
- הרשות לקבצים מנוהלות באמצעות chmod .
- מחיקה באמצעות rm (ניתן גם להוסיף -f).
- כדי להריץ במחברת של ssh צריך להוסיף סימן קריאה לפני .

### **מבוא לשפת C**

תהליך הפיתוח והקמפול בשפת C נראה כר' :



#### טיפוסים:

- ישנים טיפוסים עבר char, short, int, long :integers ובעור float, double :float. עלינו להגדיר בבירור את טיפוס המשתנה. נשים לב האם אנחנו עובדים עם signed (או ברירת המחדל) או unsigned (אפשר לציין זאת). נשים לב כי למשול unsigned char מייצג מספרים בטוחה 0 עד 255, ולעומת זאת signed הוא בטוחה 128- עד 127.
- ניתן להשתמש ב-sizeof על מנת לחשב את **מאות ה-bytes bytes = 8 bits** שיש בтип/ **משתנה מסוים**.
- כאשר נשתמש בקבועים, נוכל להשתמש בסימנים מסוימים כדי לציין את הטיפוס, למשל % unsigned, f עבור float.
- **הטיפוס character** – כלתו מיוצג על ידי ערך ASCII (ASCII). כאשר נרצה להדפיס תוו, נוכל להיעזר בפורמת % כדי להדפיס את התוו, ו-% כדי להדפיס את הערך המספרית שלו. ניתן גם לבצע פעולות אРИתמטיות על התוו, כמו שהוא ערך מספרי בסופו של דבר.
- באופן כללי ניתן להדפיס באמצעות printf ב-printf("%c", character). ניתן בסיסים שונים באמצעות %x, %d, %f.
- cast – אפשר לבצע המריה בין טיפוסים: 2 / (double) = d.
- **אין בוליאני: 0 מסמן "שקר" וכל דבר שונה מ-0 מסמן "אמת"** (פרט לערך החזרה של main, הצלחה = 0, כישלון = 1)

קלט ופלט בסיסי:

• הפקת מדפסה פלט:

```
int printf(<control string>, <arg2>, <arg2>, ..., <argn>)
```

- Example  
`printf("The bill is %d\n", x); /* x == 100 */`  
 The bill is 100

```
printf("My name is %s and I am %d years old\n", "John", 30);
```

My name is John and I am 30 years old

- The % symbol introduces conversion specification
- Returns the number of characters printed

○ ישן אופציות רבות להדפסה בפורמט מעניין:

```
char c = 'A', s[] = "Blue moon!";
```

Format	Printed	Remarks
%c	"A"	field width 1 by default
%2c	" A"	field width 2, right adjusted
%-3c	"A "	field width 3, left adjusted
%s	"Blue moon!"	field width 10 by default
%3s	"Blue moon!"	more space is needed
.6s	"Blue m"	precision 6
%-11.8s	"Blue moo "	precision 8, left adjusted

```
int i = 123;
double x = 0.123456789;
```

Format	Printed	Remarks
%d	"123"	field width 3 by default
%05d	"00123"	padded with zeros
%7o	" 173"	right adjusted, octal
%-9x	"7b "	left adjusted, hexadecimal
%-9#x	"0x7b "	left adjusted, hexadecimal
%10.5f	" 0.12346"	field width 10, precision 5
%-12.5e	"1.23457e-01"	left adjusted, e-format

• הפקת מקבלת קלט:

```
int scanf(<control string>, <arg2>, <arg2>, ..., <argn>)
```

- Example  
`int a, b;`  
`scanf("%d %d", &a, &b);`  
 The text 100 200 will assign 100 to a and 200 to b
- Returns the number of successful conversions



## שפת C

### פיתוח בסיסי ב-C

#### דיבוג:

ניתן להיעזר בכלי `gdb` על מנת לדbg את הקוד. צריך לкомפל את הקוד עם הדגל `-g`. נרים את `gdb` וניתן בארגומנט את התוכנית שלנו. לאחר מכן נשים נקודות breakpoint במקומות הפקודה `breakpoint` ומספר השורה שבה נרצה למקם את ה-`breakpoint`. נרים באמצעות `run` ונגיע ל-`breakpoint`. פקודות שימושיות:

- `[list]`.
- `(print) p`.
- `(continue) c` – המשך עד ל-`breakpoint` הבא.
- `(step) s` – כניסה פנימה.
- `quit`.
- לחיצה על ENTER מבצעת שוב את הפקודה الأخيرة.

#### דיברונות:

הכתובות בזיכרון מיוצגות בבסיס 16, כדי לייצג מספרים גדולים עם מספר קטן של ספרות: כל ספרה הקסדצימלית מייצגת 4 ספרות בבסיסים ביןaries. כאשר אנו יוצרים משתנה: `50 = a` אז, בתובת כלשיי בזיכרון מ אחסנת את הערך 50. לבתוות בזיכרון נוכל לגשת באמצעות מצביע.

- האופרטור `&` נותן לנו את הכתובת של המשתנה: `a&`.
- האופרטור `*` נותן לנו להבריז על משתנה בעל מצביע: `a*int`. כדי לגשת לערך עצמו בזיכרון נבצע `a*`.

#### מערכות ומחרוזות:

- כדי לאחסן מחרוזת, משתמש **במערך של תווים**: `"HI!"` – `s = char*`.
- נוכל לגשת לכל תו במחרוזת באמצעות `[ ]` או באמצעות `(n+s)*` כדי להתקדם במערך התווים. כאשר אנו מוסיפים 1 וכותבים למשל `(1+s)*` יש כאן **אריתמטיקה של מצביעים**. אין צורך להוסיף 4 למרות ש-`s` הוא בגודל 4 בתים. הקומpileר יודע את הטיפוס, וכך ניתן לקדם ב-1 כדי לקבל את הערך הבא במערך (**לא את ה-`byte` הבא**).

#### הקצתה זיכרון ו-valgrind:

- אנו מקצים זיכרון באמצעות `malloc` ומשחררים אותו באמצעות `free`. **שגיאה נפוצה - לא לשחרר את הזיכרון!**
- כדי להשתמש ב-valgrind צריך לкомפל את הקוד עם הדגל `-g`, ולאחר מכן להריץ `valgrind` על הקוד המקומפל (עם הדגל `leak-check=full`).
- ניתן ב-valgrind גם לראות `fault` מ/לבתו בתובת בזיכרון שאין אליה גישה.

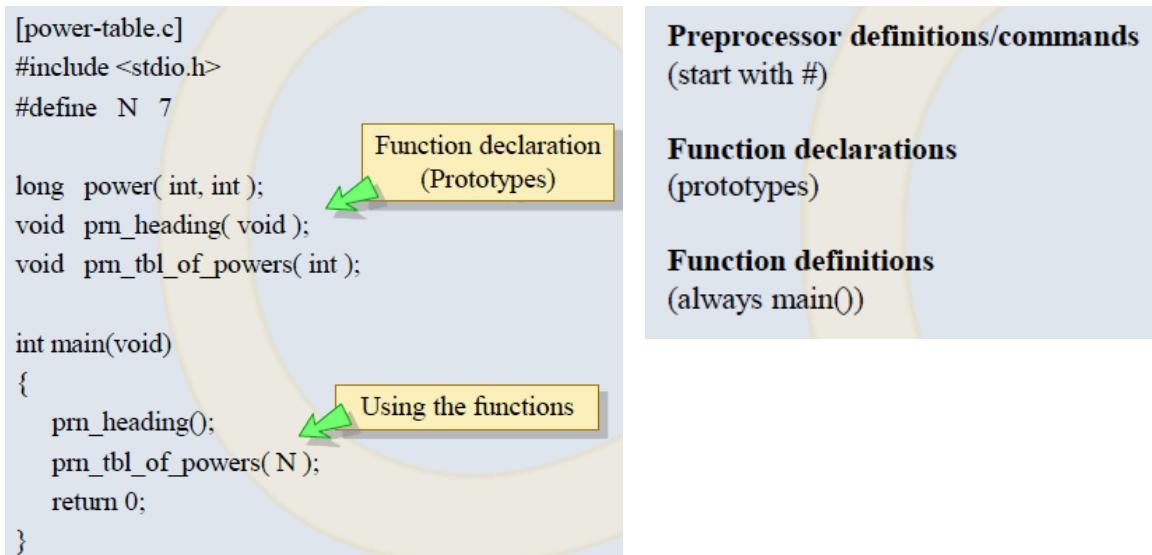
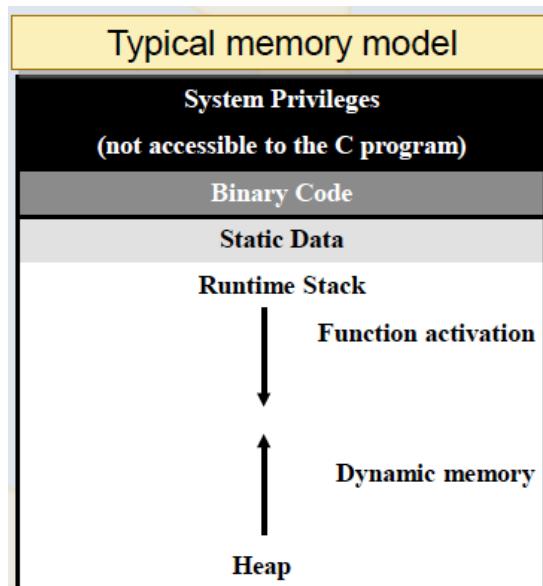
#### דברים נוספים:

- ארגומנטים בשורת הריצה: `c` הוא כמות הארגומנטים, ו-`-v` הוא מערך מחרוזות. `[0]arg[1]...` מכיל את שם התכנית.
- `atoi`: הפונקציה הזאת ממירת מ-ASCII ל-`int` **מספרים**. אם היא לא מצליחה להמיר את הקלט היא תחזיר 0.

**פונקציות****מבנה התכנית:**

פונקציה מוגדרת כפי שאנו מכירים בכל שפה: חתימה (header) וגוף הפונקציה (body) המורכב מ-declarations (הצהרות על משתנים), ו-statements (פקודות למיניהם). באשר מצהירים על פונקציה – **שמות המשתנים לא רלוונטיים, רק הטיפוס שלהם.**

מבנה תכנית כללי:

**קריאה והפעלה של פונקציות:**

- בשפת C עובדים בשיטת **call by value** (by reference). באשר אנו מעבירים משתנה int למשל כפרמטר לפונקציה, הוא מועתק לפרמטר הлокאלי שבתוך הפונקציה, **ולא משפייע על המשתנה שנשלח לפונקציה**.

- activation records** – כל ריצה של פונקציה יוצרת record שמחזיק את כל המידע הרלוונטי לצורך פעולה הפונקציה (משתנים לוקליים, state, הפרמטרים, return value, מביב runtime records) – בלחובות שמננה נקראננו. בבל records האלו מאוחסנים ב-stack, בל אחד מהם הוא frame של ריצת פונקציה בלבד.
- מודל ה-Stack** – באשר אנו מקצים זיכרון באופן דינמי, אנחנו עובדים עם ה-Stack. ה-Stack וה-Heap נמצאים באותו מרחב בתובות, וגדלים בכיוון הפוך זה זהה.
- הՔוסיה** – נshall שימוש ברקורסיה (מימוש פשוט אך משתמש במחסנית זמן הריצה) לעומת מימוש איטרטיבי (יותר יעיל אבל לעיתים מימוש מורכב).

**:Storage Classes**

- **локאלי**, ערך התחלתי זבל. הוא חי מתחילה הבלוק עד סוף הבלוק.
- **גלובלי**, אפשר לחלק משתנים כאלו בין קבצים. בקובץ אחד גדר את המשתנים לא בתוך פונקציה בלבד ( הם יהיו גלובליים, למשל a int), ובקובץ אחר נצהיר על כך שהם **שימוש במשתנה חיצוני** (extern int a ).
- **Static** (private): משתנה int static בתוך פונקציה: האתחול שלו מתבצע רק פעם אחת, והערך שלו נשמר בין קריאות לפונקציה.
- אם גדר פונקציה בתוך static int g(void); זו פונקציה שאפשר לגשת אליה רק מהקובץ בה היא הוגדרה.

**אחר:**

- פונקציות מתמטיות נמצאות בספריה math.h, צריך לקומפל עם `-lm` flag .
- לתוכר הפענוח, בודקת תנאים לוגיים: אנו צריכים לקבל מספר חיובי בטור קלט, צריך שכנתה מסוימת לא תהיה פונה וכו'. **אם התנאי לא יתקיים, התכנית תצא**. צריך לבצע `#include <assert.h>`.



## מערכות וمبرיעים

מערכות חד-מימדיות – [N], אנו תמיד צריכים לתת גודל קבוע מראש (N) למערך. אפשר לאותחל ישירות ערכים:

- `.int a[] = {2, 3, 5, -7};`
- `a[100] = {0};`
- `(null terminator char s[] = {'a', 'b', 'c', '\0'} או char s[] = "abc";`

`a[i]` is equivalent to `*(a + i)`

An array name is an address!

`p[i]` is equivalent to `*(p + i)`

The main difference between the two: a pointer can be modified, an array cannot, it is a **CONSTANT** pointer!

مبرיעים:

`int a = 1, b = 2, *p;`

**a**

**1**

**b**

**2**

**p**

→ ?

`p = &a;`

`b = *p; //is equivalent to b = a;`

**a**

**1**

**b**

**2**

**p**

→ ↓

אפשר להגדיר מבריע לבתוות בזיכרון. למשל `*k` (MBERIUT-LATON), אנו משתמשים באן-ב-`*`.

המשתנה `k` הוא מבריע, ולכן אנו יכולים לשימוש בתוכו בתובות כלשהן בזיכרון, בתובות של משתנה אחר.

כדי להשיג בתובות נוכל לבעץ `&`, להשתמש בoperator address ולקבל את הכתובת של המשתנה: `p = &a;`

כדי לקבל את הערך שיושב בכתובת עצמה, נבעץ `*` deference באמצעות `*`.

נשים לב כי `&a` מבטלים זה את זה (זה אומר לגשת למה שיש בכתובת של הכתובת של `a` = זה פשוט לגשת ל-`a`).

דוגמת שימוש:

```
#include <stdio.h>
void swap(int *p, int *q)
{
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}

int main(void)
{
    int i = 3, j = 5;
    swap(&i, &j);
    printf(" %d %d\n", i, j );
    return 0;
}
```

Define pointers as params

Use dereferenced values

Pass addresses as arguments

הערות נוספות:

- מנגנונים חשובים מאוד לשימוש בהקשר הקצת זיכרון דינמי, עובודה עם מחזוזות, והעברת פורמטרים בצורה ישלה לפונקציות.
- אפשר לאותל מצביע ל-NULL (מקביל לאותול לערך 0).
- ארכיטקטורה של מצביעים: ניתן להוסיף 1 או להחסיר 1 ממצביע ונקבל את האלמנט הבא/הקדם מאותו הטיפוס.
- מצביע גנרי: **void**. עוד לא ברור למה המצביע יצביע, לאיזה טיפוס.
- – לא ניתן לשנות את המשתנה: **const**

`const int *p => p points to a const variable`

`int * const p => p is const and cannot be changed`

`const int * const p => both.`

מחזוזותמבוא:

```
char *strcat( char *s1, const char *s2 )
{
    char *p = s1;

    while ( *p )
        ++p;

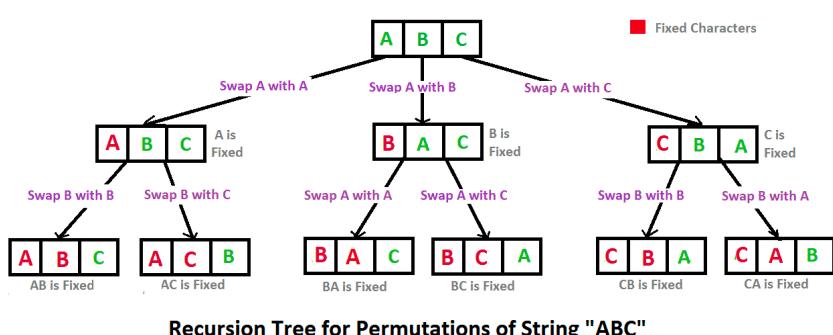
    while ( *p++ = *s2++ );

    return s1;
}
```

- string בשפת C הוא **מערך של char** שנגמר בתו מיוחד שנקרא null terminator והוא '\0'.
- יש פונקציות שימושיות רבות ב-h-string.h: strcat, strcmp, strcpy, strlen
- מימוש אפשרי ל-strcat:
  - ראשית אנו מוחכים עד ש-p מצביע ל-null terminator
  - של סיום המחרוזת, ואז ערכו ייה 0 (שקר) ונסימן.
  - אנו מפרשים unary operators לפני binary, ולכן, לכן בוללתה ה-while הזאת: ייקח את הערך הנוכחי של s2 מצביע ונשים אותו ב-p. לאחר מכן, נקדם את שנו המצביעים ב-1 (++ קורה לאחר סיום ההשמה). נצא מהלולה עד ש-s2 מגיע ל-null terminator וזה הוא יהיה שווה ל-0 (שקר).

– איך נapis את כל הפתרונות של מחזוזות מסוימת באופן רקורסיבי? נבצע swap וכייננס לrekursiva, במעלה הרקורסיה בחזרה (backtracking)

```
void permute( char *a, int l, int r ) {
    int i;
    if (l == r)
        printf("%s\n", a);
    else {
        for (i = l; i <= r; i++) {
            swap( (a+l), (a+i) );
            permute( a, l+1, r );
            swap( (a+l), (a+i) ); //backtrack
        }
    }
}
```



Recursion Tree for Permutations of String "ABC"

הקצתת זיכרון דינמית:

```

int main(void)
{
    char *w[N];           an array of pointers
    char word[MAXWORD];   work space
    int n = 0, i = 0;

    for ( i = 0; scanf("%s", word) == 1; ++i )
    {
        if ( i >= N ) {
            printf( "Sorry, at most %d words can be sorted.", N );
            exit(1);
        }
        w[i] = (char *)calloc(strlen(word) + 1, sizeof(char));
        assert(w[i]!=NULL);   Check allocation
        strcpy(w[i], word);
    }
}

```

- כל הפונקציות נמצאות בספריה `stdlib.h`. יש מספר פונקציות: `malloc`, `realloc`, `free`.
- מעדיף להשתמש ב-`calloc` כיון שהוא **מאתחלת את כל הבטים ל-0**.
  - `ch` – בדמות האלמנטים.
  - `size` – בדמות הבטים של כל אלמנט.
- הפונקציות עובדות עם `*void` וצריך להמיר לтипוס הרלוונטי.
- הן מחזירות **TOTAL בכישלון**, علينا לבדוק את ערך החזרה.
- ראיינו דוגמה למיון לקסיקוגרפי של מחחרחות: `W` הוא מערך של מצביעים, כל מצביע זה הוא מצביע למחחרחות. אנחנו מקבלים מילה, מלאקרים מקום בזיכרון בהתאם לאורך המילה – `strlen`, ועוד מעティקים. כדי למיין את המחרחות ניעזר ב-`strcmp` וביצע `swap`.

הערות נוספת:

- **ารגומנטים ל-`main`:** הארגומנטים נמצאים ב-`arg[0]` ו-`arg[1]`... `arg[n]`. תמיד שם התוכנית – `[0]`.
- **מערכים רב מימדיים:** אפשר להגדיר מערך דו-מימדי באמצעות `a[3][2] = {{1,2},{3,4},{5,6}}`.
- **אריתמטיקה עם הכתובות החזות-המערן:**
  - כאשר אנחנו נעדרים ב-`[ ]`, הוא מבצע את החישוב בעצמו וקובץ לכתובה המתואימה. לאחר מכן אם נבצע מצביע זהה נאלץ זיכרנו לח' שלמים מסווג `int`.
  - איך נאלץ זיכרנו לערך דו-מימדי? ניעזר במצביע למצביע `**a`; `**a = int`.
- **העברת פונקציה כפרמטר לפונקציה:** השתמש בתחריר הבא: `(double)(double)(*f)(*)` עברו פונקציה שמקבלת `double` ומחזירה `double`. מדובר במצביע לפונקציה. דוגמה לשימוש בפונקציה כפרמטר היא **qsort** אשר מקבל פונקציית `compare`, כדי שנוכל לספק השוואות שונות לפונקציה.
- **נשים לב** שכאשנו מבצעים `compare` ומשווים בין `*int`, `void`, **צריך להמיר אותם ל-`*int` כדי לעבוד איתם.**

```

// qsort an array of ints
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#define KEYSIZE 16

```

```

int compare_int(const void *p1, const void *p2);
void print_array(char *title, int *key, int n_elem);

```

```

int main(void)
{
    int key[] = { 4, 3, 1, 67, 55, 8, 0, 4, -5, 37, 7, 4, 2, 9, 1, -1 };
    print_array("before ", key, KEYSIZE);
    qsort(key, KEYSIZE, sizeof(int), compare_int);
    print_array("after ", key, KEYSIZE); return 0;
}

```

```

int compare_int(const void *p1, const void *p2)
{
    const int *q1 = p1, *q2 = p2;

    return ((*q1) - (*q2));
}

```



## המשר שפט C

### מבנהים

#### מבנהים (structs)

- מבנה (struct) ב-C הוא כמו class מכוון, הוא יכול להגדיר ישות עם שדות מטיפוסים שונים (אך ללא מתודות, ירושה וכו'). כדי להגדיר שם עבור struct מסוים ולא לבתוב כל פעם struct בעת שימוש, נוכל להשתמש ב-typedef על מנת להגדיר טיפוס חדש שהוא struct.
- אנחנו לעבוד עם מצביעים ל-struct, ואז אנחנו נוכל להשתמש בחז: → שמאפשר לעשיות dereference ו לגשת לשדה באמצעות נקודה, חוסף לנו את השימוש ב-(p.\*). נוכל לעשיות (p->grade).

```
struct student tmp, *p = &tmp;
```

```
tmp.grade = 'A';
tmp.last_name = "Casanova";
tmp.student_id = 9100017;
```

Expression	Equivalent Expression	Value
tmp.grade	p → grade	A
tmp.last_name	p → last_name	Casanova
( *p ).student_id	p → student_id	9100017
*p → last_name + 1	( *( p → last_name ) ) + 1	D
*( p → last_name + 2 )	( p → last_name )[ 2 ]	s

- בשנבייר struct לפונקציה נעביר מצביע אליו.

#### רישומות הקשורות:

- נגיד struct של node שמכיל מצביע next (כפי שאנו מכירים).
- בפרויקט קוד גדול נרצה לעבוד עם מספר קבועים, שצרכיהם ממשק משותף – header file עם סימת.h. שמכיל את כל ה-.list.h.includes, defines, typedefs, extern variables

## in the file: “list.h”

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

typedef char DATA; We will use chars in the example

struct linked_list
{
    DATA d;
    struct linked_list *next;
};

typedef struct linked_list ELEMENT;
typedef ELEMENT* LINK;
```

```
LINK string_to_list( )
{
    LINK head = NULL;
```

stopping criteria

```
if ((s=getchar())!=EOF)
    return NULL;
```

```
head = ( ELEMENT* )malloc( sizeof( ELEMENT ) );
assert( head!=NULL );
head→d = s
head→next = string_to_list( );
return head;
}
```

count a list recursively

```
int count( LINK head )
{
    if ( head == NULL )
        return 0;
    else
        return( 1 + count( head→next ) );
}
```

count a list iteratively

```
int count_it( LINK head )
{
    int cnt = 0;
    for ( ; head !=NULL; head = head→next )
        ++cnt;
    return cnt;
}
```



## the Preprocessor

ה-preprocessor הוא שכבה חשובה ב-C שלא קיימת בשפות אחרות, ומאפשר לבצע פעולות נוספות בקוד. כל הפקודות בשכבה זו מתחילה ב-#, והן הנחיתות לשלב שコーヘ לפני הקומpileציה. הפקודות הבסיסיות הן `#include`, `#define`.

אפשר להגדיר Macro: מעין פונקציה inline קצרה. כדי לבטא שפעולות מתמטיות יבוצעו בצורה הנכונה, **נוטף כל ארגומנט בסוגרים מייל**, כדי לא לגרום לטעויות. העורת לגבי macro:

```
#define SQ(x) ((x) * (x))
```

SQ(7 + w)      **expands to**       $((7 + w) * (7 + w))$

SQ(SQ(\*p))      **expands to**       $(((*p) * (*p))) * (((*p) *$   
 $(*p))))$

- צריך להימנע מ-; בסוף, זה לא פקודה. פשוט מtbodycut and paste
- זה לא פונקציה.

### Macros are not functions

Even when written with all the parentheses

```
#define MAX(a,b) (((a)>(b)) ? (a) : (b))
int a[4] = { 0 };
int biggest = 0;

biggest = x[0];
i = 1;
while (i < 4)
    biggest = MAX(biggest, x[i++]);
```

This code would have worked  
fine if MAX was a function

כדי לראות את תוצאת ה-preprocessor נוכל לкомPILE עם ה-flag הבא:  
`.gcc -E program.c`

#ifdef, #ifndef: conditional compilation

```
#ifndef __HEADER_H
#define __HEADER_H

.

.

.

#endif
```

**עבודה עם קבצים****פונקציות בסיסיות:**

```
void double_space( FILE *ifp, FILE *ofp )
{
    int c = 0;

    while ( ( c = fgetc( ifp ) ) != EOF )
    {
        fputc( c, ofp );
        if ( c == '\n' )
            fputc( '\n' , ofp ), found a new line, duplicate it
    }
}
```

- fopen – מקבלת שם של קובץ, ו-mode. לפתיחת (קריאה/כתיבה) ומחזירה מצביע לקובץ.
- fclose – מקבלת מצביע לקובץ שפתחנו וסגורת אותו.
- fgetc – מקבלת תו מהקובץ. היא מחזירה int (קוד ה-ASCII של התו הבא). אם ישנה שגיאה, נקבל EOF.
- fpgetc – כתבתתתו לקובץ.

ראינו דוגמה שמבצעת double-space לקובץ. היא מקבלת כבר שני מצביעים לקובצים, אחד הוא ה-input שלנו והשני הוא ה-output שלנו.

**עבודה עם IO formatted**

```
#include <stdio.h>
int main( void )
{
    int a = 0, sum = 0;
    FILE *ifp = NULL, *ofp = NULL;
    ifp = fopen( "my_file", "r" );
    ofp = fopen( "outfile", "w" );
    .....
    while ( fscanf( ifp, "%d", &a ) == 1 )
        sum += a;
    fprintf( ofp, "The sum is %d. \n", sum );
    .....
    fclose( ifp );
    fclose( ofp );
}
```

- אנחנו יכולים להשתמש בפונקציות fprintf, fscanf כדי לקרוא ולבתוב .formatted strings
- יש לנו שלושה זרים קבועים stdin, stdout, stderr. בפועל printf, scanf, fscanf(stdout) הוא .stdin קרוא מה-stdin.

**קבצים בינאריים:**

- שאנו חנכו עוסקים ב-data, fread, fwrite יש לנו אפשרות לקרוא ולבתוב בתים: לשם כך .fread, fwrite

```
while ( ( n = fread( mybuf, sizeof(char), BUFSIZE ,ifd) ) > 0 )
{
    for ( p = mybuf; p - mybuf < n; ++p )
    {
        if (islower(*p))      *p = toupper(*p);
        else if (isupper(*p)) *p = tolower(*p);
    }
    fwrite(mybuf, sizeof(char),n,ofd);
}
fclose(ifd);
fclose(ofd);
return 0;
}
```

read from the file

write to the file

close the files



- אפשר לשוטט בקבצים ולהגיע למיקומים מסוימים: .fseek, rewind, ftell

## Write a File Backwards

```

fseek( ifp, 0, SEEK_END );           move to the end of the file
fseek( ifp, -1, SEEK_CUR );         back up one character
while ( ftell( ifp ) >= 0 )
{
    c = fgetc( ifp );             move ahead one character
    putchar( c );
    fseek( ifp, -2, SEEK_CUR );   back up two characters
}
return 0;
}

```

### Bitwise Operators

פעולות:

- פעולות בוליאניות – and, or, not, שעובדות ביט מול ביט בין המשתנים הרלוונטיים.
- פעולה הזזה (shift) – תזזה שמאל: **להוסיף 0 מימין פעם אחת, שקול להכפלת 2.**
- תזזה ימינה: חלק ב-2, נרף ב-1-ים מצד שמאל כדי שהמספר ישאר חיובי (ביצוג משלים ל-2).
- אם המספר unsigned אז בשמדוים ימינה נרף ב-0-ים.

Expression	Representation	Value
a	00000000 00000000 10000010 00110101	33333
b	11111111 11111110 11010000 00101111	-77777
a & b	00000000 00000000 10000000 00100101	32805
a ^ b	11111111 11111110 01010010 00011010	-110054
a   b	11111111 11111110 11010010 00111111	-77249
~( a   b )	00000000 00000001 00101101 11000000	77248
~ a & ~ b	00000000 00000001 00101101 11000000	77248

Expression	Representation	Action
c	00000000 00000000 00000000 01011010	unshifted
c << 4	00000000 00000000 00000101 10100000	left shifted 4
a	10000000 00000000 00000000 00000000	unshifted
a >> 3	11110000 00000000 00000000 00000000	right shifted 3
b	10000000 00000000 00000000 00000000	unshifted
b >> 3	00010000 00000000 00000000 00000000	right shifted 3

### : (Packing 32 Flags)

- נתחל משתנה עבור 32 ביטים שמהווים דגלים. נניח שקיבלנו אינדקס של דגל מסוים.
- set – ניקח את המספר 1 ונעשה לו shift left על הדגל ה-0 ונרף באפסים. נבצע על זה bitwise or עם המשתנה שלם, וכך הדגל יהפוך ל-1.
- unset – ניקח את המספר 1 וכן אותו שמאליה גם פעים וنبצע על זה NOT כדי להפוך אותו ל-0. בשנעשה על זה bitwise and.
- fetch – שוב נציג את המספר 1 שמאליה, נעשה bitwise and ובודוק האם קיבלנו ערך שונה מ-0.



## ממשק בין C ל-Python

### הפצת קוד C כמודול python

רקע:

יש מספר דרכים להרחיב את הפונקציונליות של python (באמצעות imports למשל). אחת מהן, היא לכתב מודול ב-C/C++. זה יכול לשפר את הביצועים ולאפשר גישה טובה יותר לפונקציות ייבה של C ו-system calls. המטרות:

- לקרוא לפונקציות C מתוך קוד python.
- להעביר ארגומנטים מ-`python`-ל-C ולפרסר אותם בהתאם.
- לזרוק exceptions מ-`C` וליצור custom python exceptions ב-`C`.
- להגדיר משתנים גלובליים ב-`C` ולאפשר גישה אליהם מ-`python`.

הרחבה ל-C:

כדי לכתב מודולים של `python` ב-`C`, ניעזר ב-API Python שמאפשר ל-`python` לקרוא לפונקציות של `C`. הכל ארוץ לנו בתחום קובץ `Python.h` בשם header.

שלב	תיאור
1 – יצירה מודול C	<p>יצור מודול חדש ב-<code>C</code>, רצוי עם שם שונה מאשר הקוד ב-<code>C</code> כדי שנוכל לבדוק להבדין ביניהם. בדוגמה שלנו נקרא לו <code>geo.c</code> ונוסף אליו את שתי ההגדרות הבאות:</p> <pre>#define PY_SSIZE_T_CLEAN #include &lt;Python.h&gt;</pre>
2 – עטיפת הפונקציה	<p>נעטוף את הפונקציה <code>c_geo</code> שלנו כדי שנוכל לקרוא לה מ-<code>python</code>. הקוד כולל מספר אובייקטים חדשים: <code>()</code>, <code>Py_BuildValue()</code>, <code>PyArg_ParseTuple()</code>, <code>PyObject</code>, <code>PyArg_ParseTuple()</code>. פיתונו שאנו חנכו בעבר ב-<code>C</code>. השני מפרסר את הארגומנטים שאנו חנכנו ממקבילים מ-<code>python</code>, והשלישי ממיר מ-<code>double</code> של <code>python</code> למשל, מחזיר את התשובה ל-<code>python</code>. Py: מקבלת מחזורת עבור הטיפוס הרלוונטי, במקומות שימוש בפונקציות ספציפיות כמו <code>.PyFloat_FromDouble</code></p> <pre>static PyObject* geo_sum(PyObject *self, PyObject *args) {     double z;     int n;     /* This parses the Python arguments into a double (d) variable named z and int (i) variable named n */     if(!PyArg_ParseTuple(args, "di", &amp;z, &amp;n)) {         return NULL; /* In the CPython API, a NULL value is never valid for a                        PyObject* so it is used to signal that an error has occurred. */     }      /* This builds the answer ("d" = Convert a C double to a Python floating point number) back into a python object */     return Py_BuildValue("d", geo_c(z, n)); /* Py_BuildValue(...) returns a PyObject* */ }</pre>
PyMethodDef – 3	<p>ה-<code>struct</code> שמוביל מידע על פונקציה שאנו חשים לחושף ל-<code>python</code>. לחוב נגידיר <b>מערך</b> זהה, כדי ליזא מספר פונקציות שונות. המידע כולל את שם הפונקציה שתיקרא ב-<code>python</code>, את ה-<code>wrapper</code>, את ה-<code>PyModuleDef</code>, ועוד מספר ארגומנטים.</p>
PyModuleDef – 4	<p>ማתחל את המודול ש-<code>python</code> רואה: מכיל את שם המודול, את מערכ הפונקציות מהשלב הקודם וכו'.</p>
PyMODINIT_FUNC – 5	<p>זה ה-<code>entrypoint</code> ש-<code>python</code> קורא לו בשעותים <code>import</code> למודול מ-<code>C</code>. נקבע על השם של הפונקציה <code>(PyInit_module()</code> כאשר <code>module</code> זה השם שכתבנו בשלב הקודם. זו הפונקציה היחידה שאינה סטטית בכל הקובץ הזה.</p>
6 – בניה	<p>נצטרך להוציא קובץ בשם <code>setup.py</code> על מנת להתקין את הקוד שלנו. שם אנו מגדרים את שם המודול ש-<code>python</code> רואה (משלב 4, <code>PyModuleDef</code>), ואת קבצי ה-<code>source</code> (משלב 1).</p>
7 – התקינה	<p>כדי להתקין את המודול נריצ'ן:  <code>python3 setup.py build_ext -inplace</code></p>
8 – הריצה	<p>ניתן בעת לבצע <code>import</code> למודול ולהשתמש בו ☺</p>



דוגמה לכל הגדירות הנדרשות, ולশמות החוזרים שצורך לשימושם לב אליהם:

```
#define PY_SSIZE_T_CLEAN
#include <Python.h>
#include <math.h>

double geo_c(double z, int n)
{
    double geo_sum = 0;
    int i;
    for (i=0; i<n; i++) {
        geo_sum += pow(z,i);
    }
    return geo_sum;
}

static PyObject* geo_sum(PyObject *self, PyObject *args)
{
    double z;
    int n;
    if(!PyArg_ParseTuple(args, "di", &z, &n)) {
        return NULL;
    }

    return Py_BuildValue("d", geo_c(z, n));
}

static PyMethodDef geoMethods [] = {
    {"geo_sum",
     (PyCFunction) geo_sum,
     METH_VARARGS,
     PyDoc_STR("A geometric series up to n. sum_up_to_n(z^n)"),
     {NULL, NULL, 0, NULL}
};

static struct PyModuleDef geomodule = {
    PyModuleDef_HEAD_INIT,
    "geo_capi",
    NULL,
    1,
    geoMethods
};

PyMODINIT_FUNC PyInit_geo_capi(void)
{
    PyObject *m;
    m = PyModule_Create(&geomodule)
    if (!m) {
        return NULL;
    }
    return m;
}
```

**דוגמה 2 (עבודה עם מספר קבועים):**

אם יש לנו קוד ביוטר מודול C אחד, אנחנו יכולים לתמוך בעובדה מול מספר קבועים. למשל, יש לנו את `c_geo.geo` בקובץ `c_geo.c`, ואת הוצאה על הפונקציה בקובץ `cap.h`.

- נשים לב כי בדוגמה הראשונית העתקנו את הקוד של הפונקציה `c_geo` לתוכן המודול החדש `geomodule.c`. כאן אנחנו לא רוצים לעשות זאת. כך נוכל לקרוא לאותו הקוד משני ממשקים שונים, מ-C ומ-Python.
- עבשוו באשר נזכיר את `c_geo.geomodule`, נבצע "#include "cap.h"" ונקבל לייצא את הקוד בily להעתיק אותו.
- הכל נשאר אותו דבר, פרט לכך שקובץ `setup.py` נמצא בשני קבועים:

```
from setuptools import Extension, setup

module = Extension("capi_demo1", sources=['geo.c', 'geomodule.c'])
setup(name='capi_demo1',
      version='1.0',
      description='Python wrapper for custom C extension',
      ext_modules=[module])
```

**דוגמה 3 (העברה ארגומנטים מורכבים):**

נסתכל על חישוב ערך, שהקלט הוא רשימה של מספרים. `sums_ifactorial` מקבל **מערך** של שלמים. `sums_ifactorial` מקבלת מהירות של מספרים. איך נעביר רשימה כารוגמנט?

- **רישמה מ-Python ל-C:** עבור `sums_ifactorial` נגידר שני אובייקטים מטיבוס PyObject. נבצע `PyArg_ParseTuple` עם "O" כדי לקבל את האובייקט ללא שם `PyObject_Length`. parsing PyObject Length. `PyList_GetItem` מוחזר לנו את אלמנט מסויים מהרשימה.

```
// wrapper function for ifactorial_sum
static PyObject *DemoLib_iFactorialSum(PyObject *self, PyObject *args) {
    PyObject *lst;
    PyObject *item;
    long num;
    if (!PyArg_ParseTuple(args, "O", &lst)) {
        return NULL;
    }

    int n = PyObject_Length(lst);
    if (n < 0) {
        return NULL;
    }

    long *nums = (long *)malloc(n * sizeof(long));
    if (nums == NULL) {
        printf("Memory allocation failed. Exiting.\n");
        return NULL;
    }
    int i;
    for (i = 0; i < n; i++) {
        item = PyList_GetItem(lst, i);
        num = PyLong_AsLong(item);
        nums[i] = num;
    }

    unsigned long fact_sum;
    fact_sum = ifactorial_sum(nums, n);
    free(nums);
    return Py_BuildValue("i", fact_sum);
}
```



- מערך מ-C ל-Python: נגדיר את הפונקציה `GetList`, נקצת רשיימה חדשה באמצעות `New`, ונשים בה ערכיהם באמצעות `.PyList_SetItem`.

```
static PyObject* GetList(PyObject* self, PyObject* args)
{
    int N,r;
    PyObject* python_val;
    PyObject* python_int;
    if (!PyArg_ParseTuple(args, "i", &N)) {
        return NULL;
    }
    python_val = PyList_New(N);
    for (int i = 0; i < N; ++i)
    {
        r = i;
        python_int = Py_BuildValue("i", r);
        PyList_SetItem(python_val, i, python_int);
    }
    return python_val;
}
```

## יכולות נוספות

### :exceptions

על אף שאין לנו ב-C API exceptions מאפשר לנו לזרוק Python exceptions. למשל, באמצעות `PyErr_SetString` הArgument הראשון הוא סוג השגיאה, והשני הוא הודעה השגיאה.

```
int N,r;
PyObject* python_val;
PyObject* python_int;
if (!PyArg_ParseTuple(args, "i", &N)) {
    return NULL;
}
if (N < 3) {
    PyErr_SetString(PyExc_ValueError, "List length must be greater than 3");
    return NULL;
}
```

### :constants

כדי להגדיר קבועים ניעזר ב-`PyModule_AddIntConstant`:

```
PyMODINIT_FUNC PyInit_demo(void) {
    /* Assign module value */
    PyObject *module = PyModule_Create(&DemoLib_Module);

    /* Add int constant by name */
    PyModule_AddIntConstant(module, "FPUTS_FLAG", 64);

    return module;
}
```

### :debugging

כדי לדבג, אנחנו נגדיר ב-JSON את הריצת python וبنוסף את attach.(gdb)

- ראשית, נרץ את python debug console וב-`os.getpid()` כדי למצוא את ה-pid.
- עבשוו נלך לקובץ ה-C שלו ונריצו את attach.(gdb). ניתן לו את ה-pid ועוד יוכל לkopacz לשם ולדבג את ה-C.



## Python ב-Data Science – 2

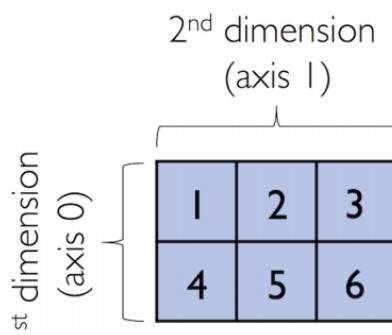
### NumPy

#### מבוא

עוקבנת היא ספריה מרכזית לביצוע חישובים מתמטיים ב-Python. היא מספקת אובייקט שהוא **מערך רב-מימדי**, ומספקת כלים לעובדה עם מערכים בלבד. היא מוממשת בעיקר ב-C ומשתמשת ב-Python בתור ממושך.

- המרכיבים של עוקבנת הם בלוקים רציפים בזיכרון והם מאוחסנים בזיכרון ייעילה cache על ידי ה-CPU. לעומת זאת, רשימות של Python הם מערכים של מצביעים לאובייקטים בזיכרון רנדומליים בזיכרון (החישוב הוא בעלות יקרה).
- ברשימות של Python קל מאוד להוסיף ולהסיר אלמנטים. שינוי גודל המערך של עוקבנת דורש ליצור מערך חדש.
- **עוקבנת יותר אלגנטית ויעילה ביצוע חישובים מתמטיים.**

#### מערכות עם ת מימדים:



- אפשר לאתחל מערך עוקבנת מרשימה רגילה של Python.
- אם נרצה לקבל מערך דו-מימדי, נעביר רשימה של רשימות. נשים לב שהגישה באינדקס הראשון (0) היא למידה השני (בוחרים שורה), והגישה באינדקס השני (1) היא למידה הראשון (בוחרים עמודה).
- מערך numpy הוא grid של ערכים,OLUMNS מאותו הטיפוס. למערך מספר תכונות:
  - `rank` – מספר המימדים של המערך. זה השדה `ndim`.
  - `shape` – המימדים של המטריצה: ( $m, n$ ) כאשר  $n$  הוא גודל המערך בימד הראשון, ו- $m$  הוא גודל המימד השני.
  - `dtype` – טיפוס האלמנטים במערך: `int`, `float` וכו'.
  - `size` – סה"כ כמות האלמנטים ב-grid (בכל המימדים).

#### דברי בנייה מערך נפוצים:

- `ones`, `zeros` – אתחול לאפסים או לאחדים. `full` – אתחול לערך ספציפי בכל התאים.
- `eye` – מטריצת הזהות (1 על האלבסן).
- `random` – ייצור ערכים רנדומליים.

#### שליפת אלמנטים - indexing - :indexing

- בדומה ל-slicing של רשימה רגילה, ניתן לקבל רק חלק מהאיברים במערך: ניתן לחזור לפי שורות ולפי עמודות.

```
# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
# [ 5  6  7  8]
# [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
# [6 7]]
b = a[:2, 1:3]
print(b)

[[2 3]
[6 7]]
```

- נשים לב כי כאשר אנחנו חותכים, אנחנו מקבלים view לאותו המידע בדיק (כמו מצביע), ולכן אם נשנה את המידע דרך ה-slice שקיבלנו, הוא ישתנה גם במערך המקורי.
- כאשר יש לנו מערך דו-מימדי ואנחנו רוצים לקבל חתך של השורה האמצעית, אם נערבב integer index עם slices נקבל מערך מ-rank נמוך יותר (חד-מימדי). אם נשתמש רק ב-slice, נקבל חתך עם rank זהה.

```
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
row_r3 = a[[1], :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)
print(row_r2, row_r2.shape)
print(row_r3, row_r3.shape)

[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
[[5 6 7 8]] (1, 4)
```



- אופציה נוספת היא **integer array indexing**: מאפשר לבנות מערך שירוטי באמצעות המידע של מערך נתון:

```

▶ a = np.array([[1,2], [3, 4], [5, 6]])
# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))

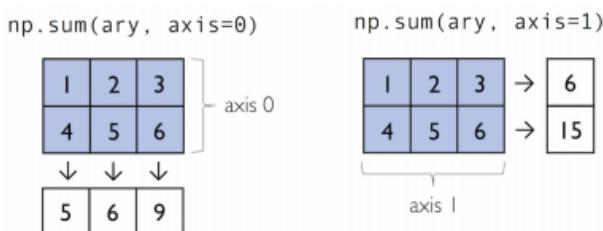
[1 4 5]
[1 4 5]

```

יש טרייך נחמד בהקשר זה, אפשר ליצור מערך של אינדקסים, ולפנוי לתת את כל השורות על ידי `np.arange()`. אופציה נוספת היא לבצע filtering באמצעות Boolean array filtering: אפשר לנו לבחור אלמנטים על פי תנאי מסוים. אפשר לחת את המערך `a` ולהציג רק את כל התנאים שעוניים על התנאי  $a > 2$ .

## פעולות נתמכות

### אריתמטיקה של מatrיצים:



- ניתן לחבר, להסח, להכפיל, ולהחלק שני מatrיצים של numpy:
- אפשר באמצעות operands או באמצעות קרייה ישירה לפונקציות `add`, `subtract`, `multiply`, `divide` ו-`divide`. נשים לב שהכפל כאן הוא לא כפָל מטריצה: הוא **כפָל של איבר-איבר**.
- אפשר להוציא שורש לכל איבר המatrיצה באמצעות `sqrt`.
- כפָל מטריצות בוצע באמצעות הפונקציה `dot` או האופרנד `@`.
- פונקציה `mean` סובכת שורות/עמודות, לפי הparameter `axis`.
- שבחר. עבור שורות ניתן `0`, עבור עמודות ניתן `1`.
- כמובן אפשר גם לבצע transpose למטריצה באמצעות השדה `T`.

### פעולות Broadcasting (התאמת מימדים):

- עוקשת מאפשר לנו לבצע עם מatrיצים אפילו אם המימדים שלהם לא מתאימים, כאשר אנו מבצעים פעולות אריתמטיות ביניהם. לעיתים יש לנו מערך קטן ומערך גדול, ואנחנו רוצים להשתמש במערך הקטן מספר פעמים כדי לבצע פעולה כלשהי על המערך הגדל.
- במקרה שאנו רוצים להוסף vector לכל שורה של מטריצה `x`. אין סיבה שנעשה זאת בלולה על כל השורות. נשים לב, כי הפעולה שકולה ליצור המטריצה `y` על ידי העמתת מספר עותקים של `v` על גבי זה (בשורות), ואז ביצוע החיבור בין `y` למטריצה `x`. אפשר לנו לבצע זאת בקלה:

```

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v # Add v to each row of x using broadcasting
print(y)

```

```

[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]

```

דוגמה נוספת לשימוש:

```

▶ # Compute outer product of vectors
v = np.array([1,2,3]) # v has shape (3,)
w = np.array([4,5]) # w has shape (2,)
# To compute an outer product, we first reshape v to be a column
# vector of shape (3, 1); we can then broadcast it against w to yield
# an output of shape (3, 2), which is the outer product of v and w:

print(np.reshape(v, (3, 1)) * w)

[[ 4  5]
 [ 8 10]
 [12 15]]

```

**פעולות נוספת:**

- numpy.linalg מספקת לנו את האפשרות לחישוב צח – חישוב מטריצה הופכית, eigvals – חישוב ערכים עצמיים. •  
 numpy.fft – התמרת פורייה. •  
 numpy.random – חישוב מספרים רנדומליים. באמצעות seed אנחנו יכולים להבטיח שהRANDOMיות תישאר קבועה. •

**ספריות נוספות:**

הספרייה SciPy: ספריה של אלגוריתמים וכלים מתמטיים לעבודה עם מערכות של numpy:

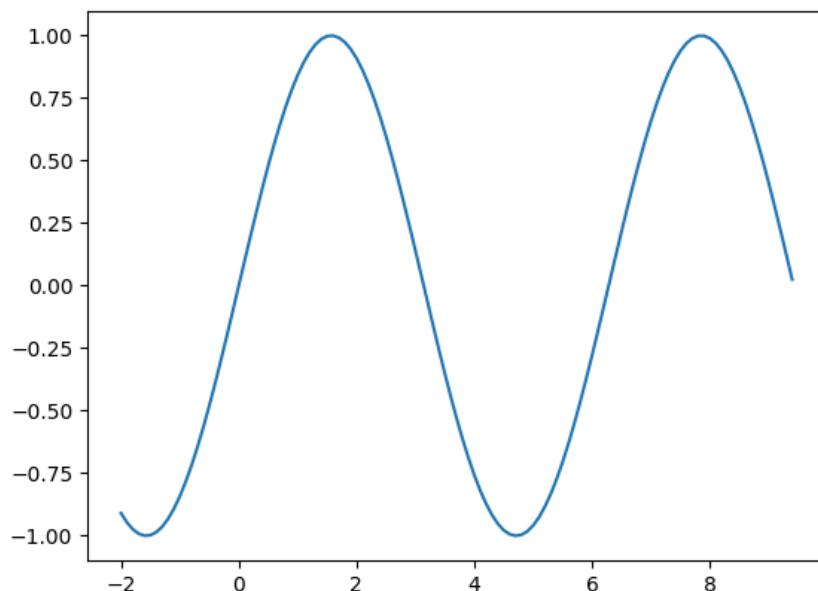
- scipy.linalg – אלגברה לינארית
- scipy.stats – סטטיסטיקה
- scipy.optimize – אופטימיזציות
- scipy.sparse – מטריצות דילולות
- scipy.signal – עיבוד אותות

הספרייה Matplotlib: מאפשרת>Create plots.

```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(-2, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
```

[<matplotlib.lines.Line2D at 0x7fb4485c8310>]





## Pandas

### מבוא

היא ספירה לביצוע **data analysis**. שני האובייקטים העיקריים הם DataFrame (טבלה) ו-Series (עמודה של מידע). היא מאפשרת לנו לקרוא מידע מקובץ csv בתוך DataFrame.

**יצירת נתונים:**

- אפשר ליצור Series על ידי האובייקט Series, כאשר נתונים לו רשימה של ערכים שיופיעו בעמודה.
- כדי ליצור DataFrame נקבע מילון המיפה את שם העמודה (מחוץ) לתוכן שלה (רשימה של ערכים). אם יש חוסר התאמה באורכים, ערכים חסרים מקבלים NaN.

```
[ ] city_names = pd.Series(['San Francisco', 'San Jose', 'Sacramento'])
population = pd.Series([852469, 485199])
pd.DataFrame({'City name': city_names, 'Population': population})
```

	City name	Population
0	San Francisco	852469.0
1	San Jose	485199.0
2	Sacramento	NaN

- הוא טבלה, המכילה רשומות כאשר כל רשומה מיוצגת בשורה נפרדת. **כל רשומה יש את האינדקס שלה**, כאשר ב-default הוא ערך מספרי עולה מ-0. לעיתים אנחנו רוצים ליצור את ה-index בעצמנו:

```
[ ] # World Cup 22 qualifiers
results = pd.DataFrame({'Home': ['Israel', 'Israel', 'Moldova'],
                        'Away': ['Denmark', 'Scotland', 'Israel'],
                        'Result': ['0-2', '1-1', '1-4']},
                       index=['match 1', 'match 2', 'match 3'])
```

	Home	Away	Result
match 1	Israel	Denmark	0-2
match 2	Israel	Scotland	1-1
match 3	Moldova	Israel	1-4

**קריאה נתונים:**

- ניתן לקרוא את הנתונים מקובץ csv במקומם ליצור אותם בעצמו באמצעות המתודה `read_csv`. ניתן במתודה `shape` כדי להבין כמה שורות וכמה עמודות יש לנו בטבלה.
- באמצעות `head` נקבל את 5 השורות הראשונות (או מספר אחר שנקבע למתודה).
- באמצעות `describe` נקבל מידע כללי על הערכים בטבלה.

**גישה נתונים:**

- כל עמודה היא שדה של אובייקט DataFrame, ואפשר לגשת אליה בקלות: `covid.country` למשל. אפשר גם לגשת עם המחרוזת של שם העמודה: `covid['country short name']`.
- באמצעות `unique` אפשר לקבל set של הערכים בעמודה.

**:indexing**

- שתי אפשרויות, `loc`, `iloc` הם מבוססות על בר שנץ'ן קודם את השורה, ואז את העמודה.
- `iloc` – בחירה מבוססת index, בחירת השורה על פי המיקום שלה במידע. אפשר גם לגשת לשורה השנייה מהסוף כמו ברשימה כל ידי `covid.iloc[-2]` למשל.
- `loc` – בחירה מבוססת label (שם השורה), כאן משתמש בערך `index` של המידע ולא המיקום שלה. כלומר, אם נעדכן את ה-index מה-default שהוא מספר להיות למשל השדה "country" (באמצעות המתודה `set_index`), יוכל לגשת באמצעות `loc` לשורה 'Israel' על ידי נתינת ה-label 'total\_vaccinations'.



## פעולות נתמכות

### מניפולציה על נתונים:

- אפשר להוסיף עמודה חדשה בהתבסס על חישוב של עמודה אחרת:

```
▶ covid['new_calc'] = covid.daily_vaccinations / 10
```

- אפשר גם לבצע חישוב בין שתי עמודות ולקבל את התוצאה:

```
▶ covid.people_fully_vaccinated / covid.total_vaccinations
```

0	NaN
1	NaN
2	NaN
3	NaN
4	NaN
...	
9322	0.108469
9323	0.134295
9324	0.145313
9325	0.150490
9326	0.162005

Length: 9327, dtype: float64

- יש מתודה **log** שמאפשרת לבצע log על עמודה.

המתודה **value\_counts** סופרת את כמות הערכים שיש לכל רשותה בעמודה מסוימת.

```
▶ covid.country.value_counts()
```

Wales	108
Northern Ireland	108
United Kingdom	108
Scotland	108
Canada	108
...	
Cape Verde	7
Bhutan	6
Palestine	2
Bahamas	1
Laos	1

Name: country, Length: 162, dtype: int64

### :filtering

- ראינו כי אפשר לפולטר לפי שדה מסוים של רשיימה: [ ]'Israel' ]
- אפשר גם לתשאל תנאים יותר מורכבים באמצעות **and** ו-**or**.
- מתודה שימושית היא **isin**, כדי לבדוק כל שלם של אופציות לשדה מסוים בלי לבצע הרבה **or**-ים.

```
▶ covid[(covid['country'].isin(['Israel','Italy'])) & (covid['total_vaccinations'] > 7000)]
```

- המתודה **dropna** מאפשרת להסיר מידע חסר (NaN). אפשר לבקש גם לבצע זאת על עמודה ספציפית.

### :grouping and sorting

- אפשר לחשב את המשכורת הממוצעת של כל קבוצה ב-nba. לכל רשותה יש את השדות Team, Salary. נבצע את החישוב באופן הבא:

```
▶ nba_avg_salary = nba.groupby(['Team']).Salary.mean()
nba_avg_salary
```

Team	
Atlanta Hawks	4.860197e+06
Boston Celtics	4.181505e+06
Brooklyn Nets	3.501898e+06
Charlotte Hornets	5.222728e+06



- 

nocell למיין את התוצאות בסדר יורד וולראות מי השחקן שמרוויח הכי הרבה:

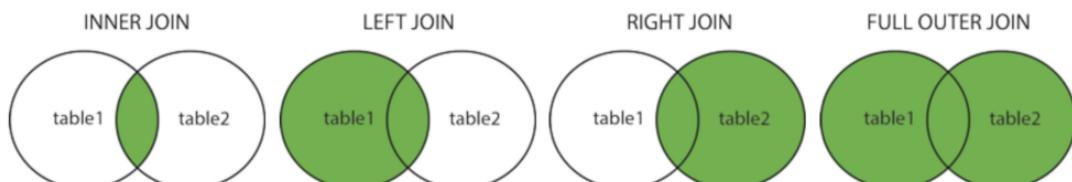
```
[ ] nba.sort_values(by='Salary', ascending=False)
```

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
109	Kobe Bryant	Los Angeles Lakers	24.0	SF	37.0	6-6	212.0	NaN	25000000.0
169	LeBron James	Cleveland Cavaliers	23.0	SF	31.0	6-8	250.0	NaN	22970500.0

- אפשר לבצע groupby גם במספר עמודות.

### :combining data

– בדומה ל-join שմבצעים ב-db, ישנו סוגים שונים.



- המתודה merge על שני DataFrames מקבילה ל-join, בולם מקבלת את החיתוך בין שתי הטבלאות, ורק את הרשומות שופיעו גם בראשונה וגם בשניה. נשים לב כי צריך לספק את הפורמטor on, לפי מה לבצע את ה-merge, למשל 'id'.

- כדי לבצע סוג שונה של join נוכל לספק את הפורמטor how, אם ניתן לו את 'left' אזי נקבל left join, כל הרשומות של הטבלה השמאלית ישארו, ורק רשומות בטבלה הימנית שיש להן חיתוך עם השמאלית יתומספו.

- אם נרצה לאחד את השורות של שתי טבלאות אחת, זו פעולהsq בשם concat.  
– ניתן להחיל על DataFrame פונקציות lambda או פונקציות של ממש.

Create a lambda function that will capitalize strings.

```
[ ] capitalizer = lambda x: x.capitalize()
```

Capitalize both Mjob and Fjob

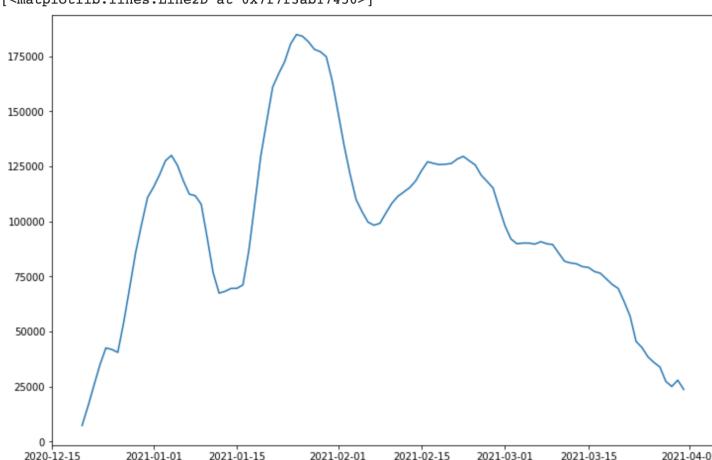
```
[ ] stud_alcoh['Mjob'] = stud_alcoh['Mjob'].apply(capitalizer)
stud_alcoh['Fjob'] = stud_alcoh['Fjob'].apply(capitalizer)
```

## Matplotlib

גם כאן נוכל להיעזר בספריה matplotlib

```
▶ plt.figure(figsize=(12,8))
plt.plot(isr_covid['date'], isr_covid['daily_vaccinations'])
```

```
❷ [matplotlib.lines.Line2D at 0x7f7f3ab17450]
```





מתודה נוספת היא subplot, המאפשרת לנו ליצור כמה גרפים במקביל.

```
[ ] # Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.figure(figsize=(12,14))

# Make the first plot
plt.subplot(3, 1, 1)
plt.plot(pd.to_datetime(covid_cases['Date'].astype(str)), covid_cases['Confirmed'])
plt.title('Confirmed Cases')

# Set the second subplot as active, and make the second plot.
plt.subplot(3, 1, 2)
plt.plot(pd.to_datetime(covid_cases['Date'].astype(str)), covid_cases['Deaths'])
plt.title('Deaths')

# Set the third subplot as active, and make the third plot.
plt.subplot(3, 1, 3)
plt.plot(pd.to_datetime(covid_cases['Date'].astype(str)), covid_cases['Active'])
plt.title('Active')

# Show the figure.
plt.show()
```

## Machine learning

### מבוא

machine learning באופן כלל הוא כל מודל שלומד באמצעות data לענות על שאלה מסוימת. יש סוגים שונים של מודלים כאשר אחד הסוגים הוא AI – מודלים של רשתות נירוכים עמוקות, למידה עמוקה.

```

import numpy as np
import matplotlib.pyplot as plt
np.random.seed(162)
# Create data
N = 25
x = np.random.rand(N)*100
y = x**2 + np.random.normal(1, 2, size=N)*100
colors = (0,0,0)
area = np.pi*3

# Plot
plt.scatter(x, y, s=area, alpha=0.5)
plt.title('House Price Vs Square')
plt.xlabel('Square')
plt.ylabel('Price')
plt.show()

```

נסתכל על דוגמה בסיסית – ביהינת N זוגות מהצורה  $(x, y)$  כאשר  $x$  הוא גודל הבית, ו- $y$  הוא המחיר. המשימה היא למצוא מחיר של בית בגודל  $x$ . בולם, להבין מהי הפונקציה  $f$  כך ש- $y = f(x)$ .

- אם ננסה לשרטט גרף, לא נוכל למצאו את הערך של  $y$  עבור למשל  $x = 60$ , הקשר אינו י לנארו – רגסיה ליניארית לא עוזרת לנו בכך. זו בעיה עם משתנה אחד בלבד.

- מה יקרה אם עברנו אונחנו יודעים גם את המשתנה  $age$ ? יש לנו שני משתנים:  $(age, square) \rightarrow price$ .

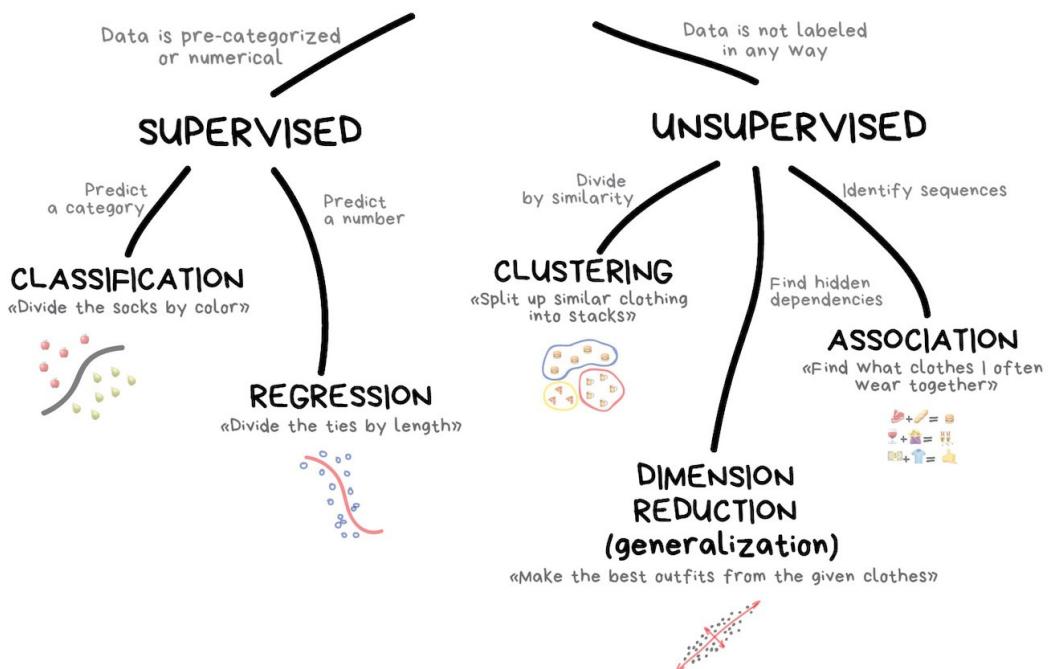
- ניתן להעלות את המימדים של הבעיה ל-d מימדיים כלשהם:  $\mathbb{R}^d \in x$ .

עולם ה-ML הקלסי מחלק לשני חלקים:

- **supervised** (למישה מודרכת) – יש לנו את הנתונים האמתיים להתבסס עליהם כולל הפתרון שלהם, התשובה הרצiosa, ומהם המודול לומד את הנוסחה. יש לנו את הפלט הנכון ומולו המודול מתאם. באנו ישן שימוש כמו classification (פלט לא מספור: דיזי קטגוריות כמו חיות, פירות), ו-regression (באשר הפלט הוא מספר ממשי: גיל, גמן, משקל).

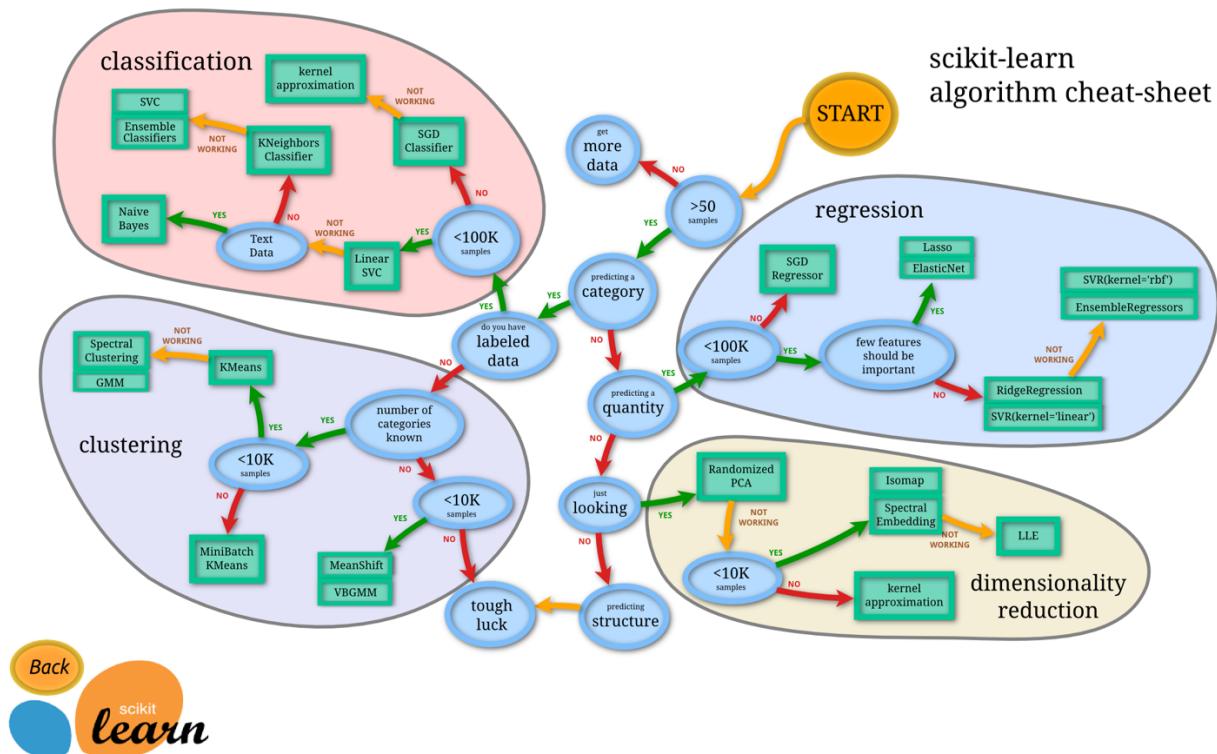
- **unsupervised** (למישה לא מודרכת) – הפלט הוא data בלבד, ללא הפלטים המתאים לכל סט נתונים. באנו ישן שימוש כמו clustering (סדרת תמונות לפי קטgorיה, הוא מוצא דמיון מסוים ומאנגד תמונות דומות תחת אותה קטgorיה. הוא לא יודע להגיד שהקטgorיה היא כלב או חתול), association (מציאת התאמה בין נתוניים) ו-generalization (הו דודת מימדים של בעיה).

## CLASSICAL MACHINE LEARNING



ניתן בספריה scikit-learn, שמאפשרת לנו לעבוד עם מודלים של ML ב-python. יש 4 שימושים בסיסיים שהוא יכול לעזור לנו בהן: regression, classification, clustering, dimension reduction. כדי לדעת באיזה מודל אנחנו צריכים להשתמש, יש דיאגרמה מקיפה. ה- pipeline של החישוב עבד כך:

- בחירת מודל.
- בחירת hyper-parameters, הפרמטרים היכי טובים למודל.
- ארגון המידע לתוך .feature matrix + target vector
- מתאימים את המודל למידע באמצעות .fit()
- חיזוי פלטים עבור מידע חדש.



## Regression

```
[ ] X = diabetes.data[:, [2]]
Y = diabetes.target

[ ] X = df_diabetes[['bmi']].to_numpy()
y = df_diabetes['target'].to_numpy()

[ ] from sklearn import linear_model
model = linear_model.LinearRegression()
model.fit(X, Y)
print(model.coef_, model.intercept_)

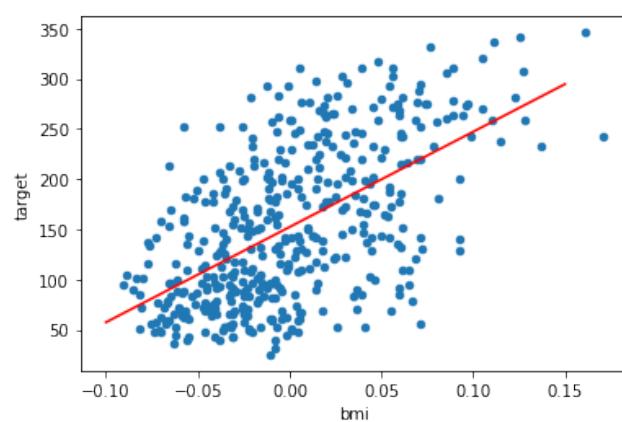
[949.43526038] 152.1334841628967

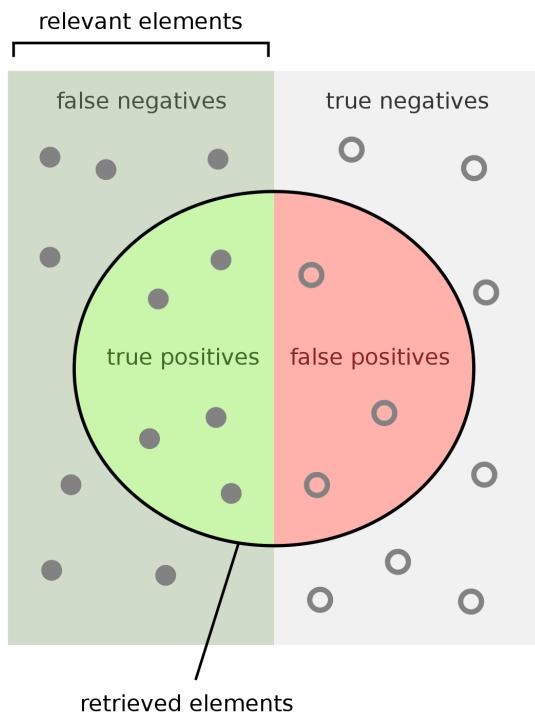
[ ] print(model.score(X, Y))

0.3439237602253803

[ ] model.predict(np.array([[0.05], [-0.02]]))
array([199.60524718, 133.14477896])
```

- ניקח dataset של חול脾 סכנתה: באמצעות `.load_diabetes()`
- 10 העמודות הראשונות מכילות מדדים בלבדם, והעמודה ה-11 היא מדד כמותי של התקדמות המחללה לאחר שנה. הקונבנצייה היא לשימוש את המשתנים מצד שמאל, ואז את ה-target מצד ימין (ניתן לגשת אליו באמצעות אינדקס -1).
- בניית **DataFrame** של `pandas`.
- באמצעות `plot` נוכל לשרטט גרפים שונים, כמו למשל target בתלות `bmi`.
- באמצעות `to_numpy` נקבל אובייקטים של `numpy`, ניקח את `X` להיות עמודת `bmi`, ו-`Y` להיות עמודת ה-`target`.
- באמצעות `import linear_model`. נקרא לפונקציה `fit` שהיא עשויה את ההליך האימון של המודל באמצעות מטריצת המשתנים, ואת המטרה: `.fit(X, Y)`.
- עבור `X` חדש (מאותו מידן) נוכל לקרוא לפונקציה `predict` כדי למצוא את ה-`Y` המתאים לפי מה שהמודול חוצה.
- לבסוף נוכל לבנות את גרפ הרגression הליינארית על הנקודות האמיטיות.





How many retrieved items are relevant?

Precision =  $\frac{\text{true positives}}{\text{true positives} + \text{false positives}}$

How many relevant items are retrieved?

Recall =  $\frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$

## Classification

נוקח dataset של פרחים: באמצעות `datasets.load_iris()`

- ישנם 3 סוגים שונים של Iris: נרצה לחזות את הסוג המתאים.
- נצבע את ה-data כמו שהוא נתון, ונראה את 3 ה-classes הקיימים.
- נשתמש במודל `train_test_split`. נחלק את ה-data לרוב - 80% אימון ו-20% לבדיקה.
- בעת ניקח את המודל `KNeighborsClassifier` עם פרמטר 5 – כולם עברו כל נקודה הוא יקח את 5 השכנים הקרובים אליה ולפי זה יסיק את הסוג. נאמן אותו על ה-data באמצעות `.predict`.
- בצע את התחזית באמצעות `.predict`.

מדדיה:

- נבדוק את הפרדיקציה אל מול התוצאה הרצiosa.
- מקובל לחלק ל-TP, FP, FN, TN – לראות מה השונות/טטיות התקן של המודל.
- מחלקים את ה-dataset לחלקים שווים (בדוגמה זו 5) ומשאירים חלק בצד לבדיקה, ומאמנים על השאר.

```
[ ] np.mean(model.predict(X_test) == y_test) # Accuracy
```

1.0

```
[ ] import sklearn.metrics as metrics
metrics.accuracy_score(model.predict(X_test), y_test)
```

1.0

```
[ ] print(metrics.classification_report(model.predict(X_test), y_test))
```

	precision	recall	f1-score	support
setosa	1.00	1.00	1.00	11
versicolor	1.00	1.00	1.00	13
virginica	1.00	1.00	1.00	6
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

למזהה מספר מסוים של קטגוריות. אפשר להשתמש באלגוריתם כמו k-means clustering – שימושי אם אין לנו labeled dataset עם הקטגוריות שאנו חוצים לחזות, אבל עדין אנו רוצים למצוא מספר מסוים של קטגוריות. אפשר להשתמש באלגוריתם כמו k-means clustering כדי לבצע clustering לנוקודות.

## Dimension Reduction (PCA)

נוקח dataset של חולי סרטן: באמצעות `datasets.load_breast_cancer()`

- הוא מכיל 30 משתנים, אשר ה-target הוא良性/Benign target.
- אנו מבצעים התאמת לפי ערך **עמודת ה-target** בצורה הבאה: `[bc.target_names[bc.target]]`.
- ננרטם את ה-data באמצעות `StandardScaler`.

בעת, ניקח את המודל `PCA`, ובוחר 2 קומפוננטות – ככלומר נרצה להוריד את מינוני הבעה מ-30 משתנים ל-2 משתנים. נקרא לפונקציה `fit_transform` על בסיס מטריצה של הנתונים.