{iT}Factory

Întâlnirea 1

Setup, Variabile, Tipuri de date

Sfaturi generale

- Să tratezi cu seriozitate și profesionalism acest nou obiectiv.
- Cei care își ating obiectivele nu sunt întotdeauna cei mai smart, dar întotdeauna vor fi cei mai muncitori!
- Alocă-ți timp pentru studiu. Rutina dă consistență. Consistența dă excelență.
- Să faci tot posibilul să participi la toate sesiunile live.
- Să lași comentarii explicative în notițele tale, informații pentru tine din viitor.
- Recomand să vizualizezi înregistrarea. Să îți notezi aspectele importante + întrebări pentru trainer pentru ora următoare.
- Să îți faci temele și unde nu reușești singur, să întrebi pe grup. Trainerul va răspunde și vor beneficia și ceilalți cursanți de răspuns.
- Poți chiar să faci un grup doar de studenți și să vă întâlniți o dată pe săptămână să discutați temele împreună. Fiecare va veni cu o perspectivă nouă și în final toți vor avea de câștigat.
- În timpul orelor, să ai curaj să pui întrebări când ceva nu e clar.

Reguli curs

- Va exista un sheet de prezență.
- În cadrul acestuia îți vei asuma și noțiunile învățate. Nu trecem mai departe până nu își asumă toți cursanții noile concepte.
- Temele se vor adăuga în Folderul grupei, unde vei crea un folder cu numele tău. Vei primi feedback la aceste teme.
- Temele vor fi împărțite în 2 categorii:
  - Obligatorii (se pot face doar cu noțiunile învățate la clasă);
  - Opționale (acestea vor fi mai avansate și necesită poate informare suplimentară). Acest lucru îi va motiva și pe cei care au mai mult timp și le place să se aventureze prin task-uri mai dificile.
- Te rog să mă întrerupi oricând ai întrebări. Doar așa îmi pot da seama unde trebuie să mai insist cu explicații/ exemple.
- Te rog să intri cu 3 minute mai devreme la curs în caz că apar probleme tehnice. Astfel putem profita la maxim de cele 2 ore alocate.
- Dacă nu poți intra, sau dacă întârzii, anunță trainerul pe grup.

Obiective principale

Până la final TOȚI veți avea:

- Cunoștințe solide despre bazele programării în Python.
- Cunoștințe mai avansate și extrem de utile despre programarea bazată pe obiecte.
- Capacitatea de a identifica elemente și de a scrie test scripts cu ajutorul Selenium.
- Un Proiect final de testare automată a aplicațiilor web:
  - Acesta va folosi tendințele actuale: metodologia Behavior Driven Development și Page Object Model Design pattern.
  - Va avea capacitatea să genereze rapoarte HTML (‘living documentation’).
  - Veți ști de la A la Z acest framework, astfel că veți avea capacitatea să continuați să dezvoltați post curs acest proiect (pentru orice website doriți).
- Noțiuni de baza despre API testing - testarea backend - ce e în spatele unui website.

* toți cei care sunt activi, implicați, își fac temele, dedica timp pentru studiu individual și pun întrebări trainerului vor atinge aceste obiective.

Obiective secundare

Nu fac parte din curricula cursului LIVE dar îți punem la dispoziție materiale extra ca să ai un avantaj la interviuri. Sfatul meu e să te concentrezi pe ele doar după cursul live. Să nu fii overwhelmed de informații noi.

- Cunoștințe ale bazelor de date relaționale - mySQL - Curs baze de date.
- Cunoștințe teoretice despre testarea manuală - acces la o platforma mobilă.
- Capacitatea de a construi un mic brand personal - Curs Portofoliu Wordpress. Trebuie să ai:
    - Website propriu prin care angajatorul să te cunoască pe tine și munca ta.
    - CV european în engleză.
    - Profil LinkedIn.
    - Github public - un loc în cloud unde pui codul scris de tine.
    - Vei primi feedback dacă ne trimiți un email cu ele la hello@itfactory.ro.

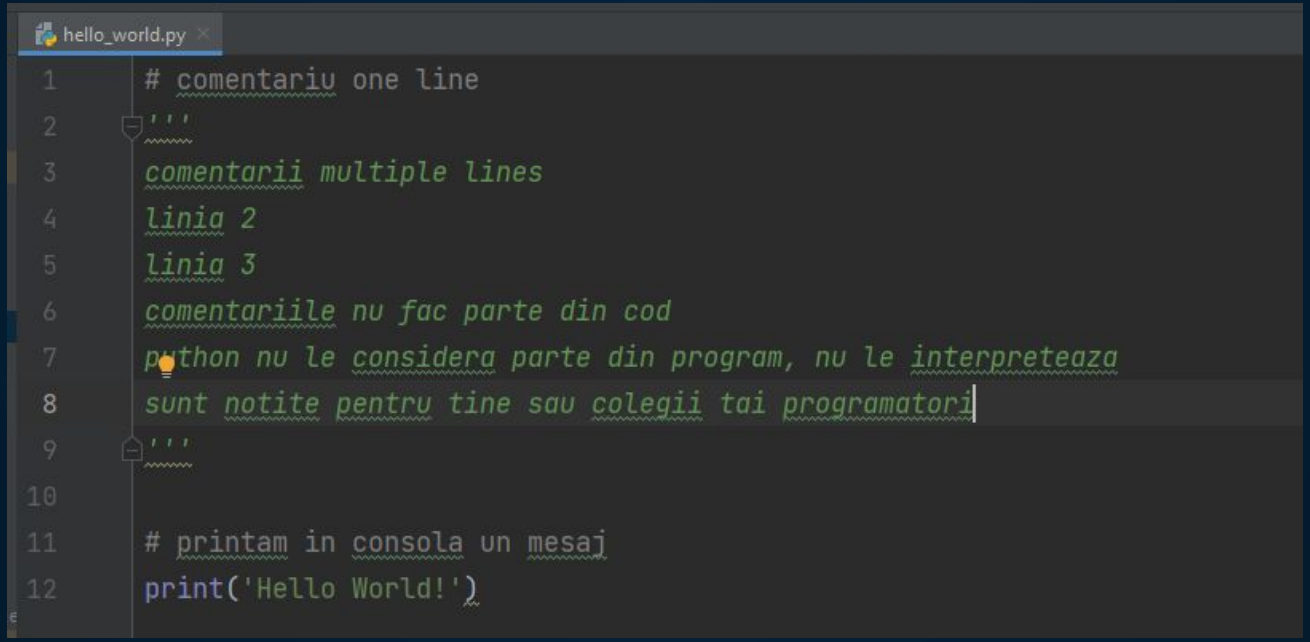
Obiective Întâlnire1

- Toată lumea să aibă set-up-ul funcțional.
- Să înțelegi cum funcționează programarea și de ce e importantă pentru Automation.
- Primul program Hello World.
- Ce este un comentariu?
- Să știi și să poți explica altora ce e o variabilă și de ce avem nevoie de ea.
- Să înțelegi cele mai uzuale tipuri de date.
- Să înțelegi ce este type casting și de ce e util.
- Să înțelegi cum funcționează print statement.
- Să știi cum iei date de la tastatură (user input).
- Ce e un assert și la ce e bun?
- Să descoperi și să aprofundezi complexitatea unui string:
    - Index
    - Length
    - Slicing
    - Metode ajutătoare

Principii de baza în programare

- A compila = a traduce din ‘human reading syntax’ în ‘machine language’
- Codul se interpretează secvențial, linie cu linie, de sus în jos
- Machine language = binary code (cod binar) - combinații diferite de 0 și 1
- Principiul seamănă cu cel din codul morse. Pt 1 se transmite un impuls electric, pt 0 o pauză.
- 1 bit = memorie în care încape doar o singură valoare. 1 (true), 0 (false)
- 1 Byte = 8 biți. Numere între 0 (00000000) și 255 (11111111)
- 1 Kilobyte = 1.024 bytes
- 1 Megabyte = 1.024 kilobytes (1.048.575 bytes)
- Terminal - zona în care trimitem instrucțiuni către program - altele decât cod python
  - Ex: ‘python – version’
  - Tot de aici putem instala librării externe (ex: pip install selenium)
- Consola - zona în care primim output (răspuns vizual) de la programul rulat
- IDE - Integrated Development Environment - Pycharm. Este un editor de cod
- Venv - Virtual environment - zona care folosesc în mod izolat și securizat toate librăriile externe

Hello World + Comentarii



Variabile

```
# am declarat si initializat variabile
marca_masina = 'Volvo'
model_masina = 'XC 60'
```

- O variabilă este un container din memorie care stochează valori.
- Îți poți imagina o cutiuță pe care punem un label.
- Variabilele au nume unic că să poată fi identificate și folosite ulterior.
- Variabila e creată în momentul în care îi atribuim o valoare.
- Nu putem pune spațiu în numele unei variabile (my_var sau myVar).
- Variabilele încep cu litera mică dar pot conține cifre (user1) și simbolul _ .
- Variabilele sunt case sensitive (myvar=3 e diferită de myVar=5).
- Variabilele pot să își schimbe valoarea pe parcursul execuției programului (suprascriere).
  - Și chiar și tipul de date
- Putem atribui mai multe valori in one line, sau aceeași valoare mai multor variabile.

```
x, y, z = "Orange", "Banana", "Cherry"
a = b = c = "Apple"
```

Tipuri de date

- Datele salvate în variabile pot avea diferite tipuri.
- Există mai multe tipuri de date, dar cele mai importante/ folosite sunt:
  - int - număr întreg;
  - float - număr zecimal;
  - bool - adevărat/fals;
  - string - șir de caractere de la tastatură delimitate de ‘ ‘ sau “ “ .
- În întâlnirea 3 vom discuta și despre colecții, care sunt toate tipurile de date (list, dict, set, tuple).

```
marca = 'Dacia' # string - sir de caractere
an_fabricatie = 1987 # int - nr intreg
pret = 2300.500 # float -  nr zecimal
inmatriculata = False # bool - A/F
```

Funcția type() și type casting

- O funcție este o logică de cod predefinită care face ceva.
- Are sintaxa nume_functie().
- În paranteze punem datele de intrare / input.
- Vom discuta pe larg despre funcții în capitolele următoare.
- Funcția type ne expune tipul de date al variabilei date că input.

```
nume = 'Andy'
print(type(nume)) # => <class 'str'>
```

- Funcțiile int(), str(), bool(), float() schimbă tipul de date. (ex: int('3') => 3)

```
cifra = '3'
cifra = int(cifra) # schimbam tipul de date / type casting
print(type(cifra)) # => <class 'int'>
```

Funcția print()

- Printează în consolă ce punem între paranteze.
- Dacă dorim să facem o concatenare (adunare) de stringuri, putem face asta cu +.

```
nume = 'Andy'
prenume = 'Sinpetrean'
print('Numele meu complet este ' + prenume + ' ' + nume)
```

- Dacă dorim să adunăm int + string (mere cu pere), vom primi eroare.
- Exista 2 soluții pentru a rezolva această problemă.

```
nume = 'Andy'
varsta = 33
print('Ma numesc ' + nume + ' si am varsta de ' + str(varsta))
print(f'Ma numesc {nume} si am varsta de {varsta}') # aceasta varianta e recomandata
```

Assert

- Assert e o modalitate de a face verificări în programare.
- Verifică dacă statement (propoziția) este evaluată în final că True.
- Dacă răspunsul e True, codul curge mai departe.
- Dacă răspunsul e False, codul se oprește și dă o eroare. Nu se execută liniile de cod ce urmează.
- Toate testele automate se termină în mod normal cu un assert, deci cu o verificare.

```
a = 1
# il intreb pe python: hey, a este egal cu 1?
assert a == 1
print('trec pe aici')
assert a == 2
print('nu mai ajung aici')
```

Funcția input()

- Funcția input() ne ajută să luăm date de la tastatură și să le salvăm într-o variabilă.
- Dacă nu facem type casting, defaultul datelor date de user = string.
- Ulterior putem accesa valorile salvate în variabile după necesitate.

```
nume = input('Alege un nume') # default - string
varsta = int(input('Alege o varsta')) # fortam varsta sa fie un int
```

String (index, len(), slicing, metode)

- Fiecare caracter dintr-un string, are un număr asociat (index), începând de la 0.
- Funcția len() ne spune câte caractere are stringul.
- Slicing - putem accesa ‘felii’ din string folosind următoarea sintaxă:
  - My_str[start_pos, end_pos, pas]
- După my_str dacă punem . ajungem la funcții ajutătoare:
  - Upper, lower, replace, count etc.
  - Accesați descrierea lor apăsând CTRL+Click pe numele lor.

```
prop = 'Andy este prescurtarea de la Andrei'
print(len(prop)) # => 35
print(prop[0]) # => A
print(prop[0:3]) # => 'And', indexul de la last_pos se exclude, pasul e optional
print(prop[::-1]) # => parcurgere inversa
print(prop.upper()) # => tot cu litere mari
```

{iT}Factory

Întrebări de interviu:

- Ce este o variabila?
- Care sunt cele mai importante tipuri de date?
- Când folosim funcția type()?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 2

Operatori, condiționale

Obiective Întâlnire 2

- Să cunoști tipurile principale de operatori:
  - De atribuire;
  - Aritmetici;
  - De comparare;
  - Logici.
- Să înțelegi cum se folosește condiționalul if else (flow control):
  - If simplu;
  - If / else;
  - If / else if / else.

Operatori de atribuire

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3

Operatori aritmetici

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y

Operatori de comparare

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Operatori logici

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

if...

- În engleză acest principiu se numește ‘flow control’ - controlăm pe unde curge codul
- Un if simplu e ca o ușă: daca ușa e deschisă (true), se va executa codul din spate. Dacă ușa (condiția) e închisă (false), python nu va afla ce e în spatele ușii. Pentru Python, acea zona de cod e inaccesibilă, nu există.
- După cele: ale unei ramuri, când apăsăm ‘Enter’ se vor pune automat 4 spații sau un TAB

Acest lucru se numește indentare. Indentarea are scopul de a-i transmite lui python de unde până unde ține blocul de cod corespunzător acelei condiții. Sau, altfel spus, marchează pereții camerei din spatele ușii.

```
nota_de_trecere = 4.5
nota = float(input('alege nota'))
if nota > nota_de_trecere:
    print(f'Ai luat nota {nota}')
    print('Felicitari ai trecut examenul!')
```

- E ok logica codului?
- Găsește ‘bug-ul’.

If... else

- Dacă (condiție) atunci (executăm codul).
- Are tot timpul fix 2 ramuri.
- If are condiție urmată de:
- Else nu mai are nevoie de condiție, deoarece înseamnă în celălat caz.
    - Ex: Un număr dacă nu e par, e automat impar.

```
# constanta - are o valoare stabila, nu ne dorim sa o schimbe nimeni
# standardul este sa o scriem cu litere mari
NOTA_DE_TRECERE = 4.5
nota = float(input('alege nota'))
if nota >= NOTA_DE_TRECERE:
    print(f'Ai luat nota {nota}')
    print('Felicitari ai trecut examenul!')
else:
    print(f'Ai luat doar nota {nota}')
    print('Ne vedem la vara! Ai picat examenul')
```

If... else if... else

- Se folosește când avem mai mult de 2 situații posibile.
- Condițiile se evaluează de sus în jos.
- Se execută codul aferent primei condiții adevărate.
- După ce s-a găsit cu true, nu se mai verifică ce a mai rămas mai jos.

```
# robotel telefonic
optiunea = int(input('alege o optiune'))
if optiunea == 0:
    print('meniu anterior')
elif optiunea == 1:
    print('ati ales ro')
elif optiunea == 2:
    print('ati ales eng')
else:
    print('nu am identificat optiunea, mai incearca')
```

- Un singur if la început.
- Oricâte elif-uri sunt necesare.
- Un singur else la final.
- Dacă nu găsește niciun true mai sus, else se vă executa automat (e ca un default).

{iT}Factory

Întrebări de interviu:

- Ce reprezintă operatorii logici?
- Ce reprezintă operatorii aritmetici?
- Când folosim condiționalul if?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 3

Structuri de date

Obiective Întâlnire 3

- Să înțelegi ce sunt, care sunt particularitățile și cum se folosesc următoarele structuri de date:
  - listă;
  - dicționar;
  - set;
  - tuplu.

List

- Listele păstrează mai multe valori într-o singura variabilă.
- În Python, putem păstra diferite tipuri de date în aceeași listă.
- Fiecare element din listă are index începând de la 0 (ca și string-ul).
- Lista este ordonată, astfel când adăugăm un element nou, acesta se va pune la final.
- Lista este mutabilă, adică putem adăuga, șterge sau schimba elemente din ea.
- În listă putem pune valori duplicate.
- len() ne va da dimensiunea listei - Câte elemente avem în listă?

```
list1 = ["abc", 34, True, 40, "male", "male"]
print(list1)
print(list1[0])
print(len(list1))
```

Dict

- Dicționarele păstrează date de tip cheie: valoare.
- Dict-urile sunt ordonate.
- Dict-urile sunt mutabile, deci valorile pot fi schimbate.
- Cheile sunt unice, nu putem avea chei duplicate, ar crea confuzie.
- Cheile sunt ca niște porecle pentru index-uri.
- Putem folosi len() pentru a afla dimensiunea.

Set

- Set-urile păstrează mai multe valori unice într-o variabilă.
- Nu sunt ordonate sau indexate.
- Datorită acestui fapt sunt și imutabile (nu putem schimba locația elementelor).
- Se pot doar adăuga sau șterge elemente.
- Putem folosi len() pentru a afla dimensiunea.

```
culori = {'alb', 'rosu', 'galben'}
print(culori)
print(len(culori))
```

Tuple

- Păstrează mai multe valori imbutabile într-o singură variabilă.
- Valorile sunt ordonate, încep de la index 0.
- Valorile sunt imutabile, odată definite, așa rămân. Nu se mai pot adăuga/șterge valori.
- Acceptă valori duplicate.
- Putem folosi len() pentru a afla dimensiunea.

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
print(len(thistuple))
```

{iT}Factory

Întrebări de interviu:

- Ce este o listă?
- Care este diferența dintre un Set și un Tuple?
- Când folosim dict?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 4

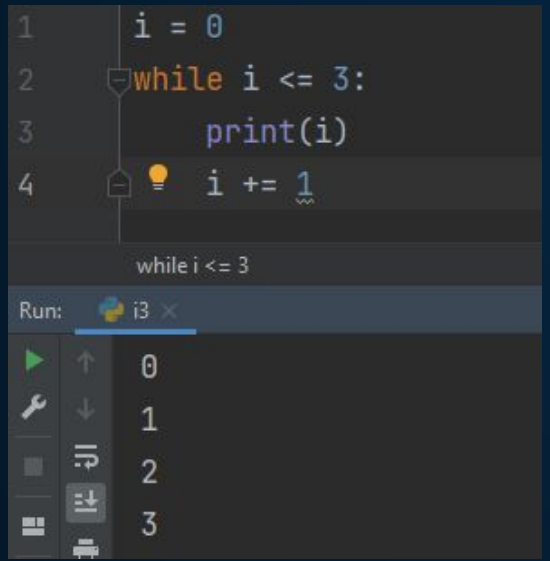
Cicluri repetitive

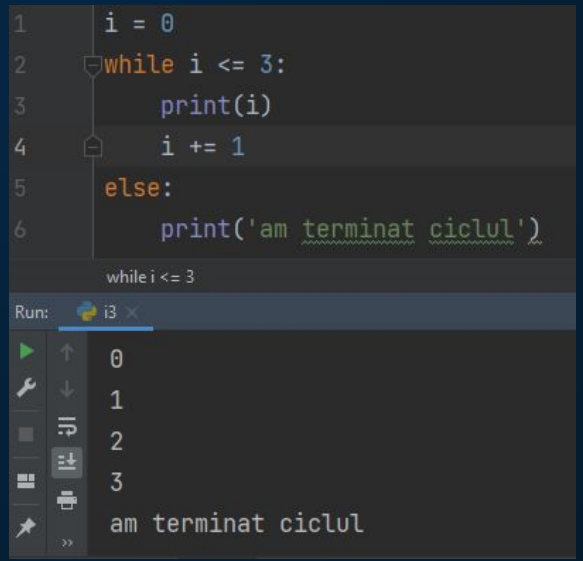
Obiective Întâlnire 4

- Să înțelegi ce sunt, care sunt particularitățile și cum se folosesc ciclurile repetitive:
  - while;
  - while else;
  - for each;
  - for;
  - for else.
- Să poți controla iterațiile cu:
  - break;
  - continue.

While / while else

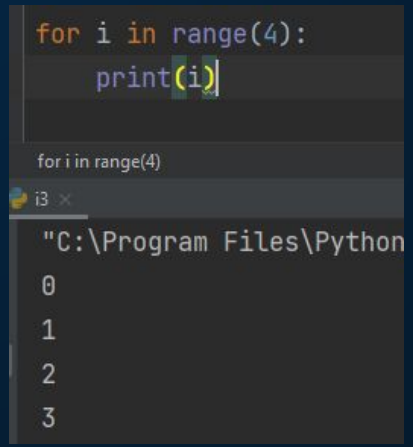
- Se execută un bloc de cod atât timp cât o condiție e adevărată.
- Opțional: la final se poate pune else, această zonă se execută o dată, la final.

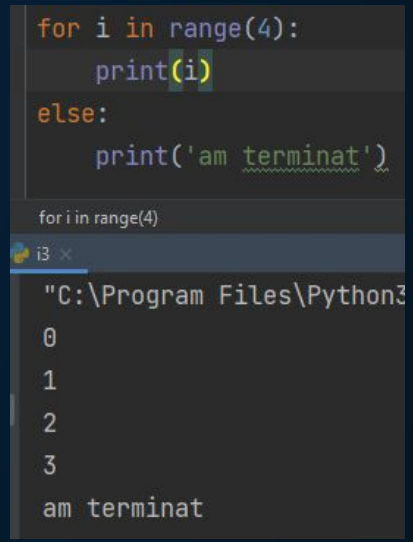




For / for else

- Se execută un bloc de cod pentru fiecare valoare din range.
- Range seamănă cu slicing. Ne spune:
  - De unde începem? Default e 0.
  - Până unde iterăm?
  - Opțional: pasul.
- Opțional: la final se poate pune else.
  - această zonă se execută o dată, la final.





For each

- Se parcurge o colecție și se salvează fiecare element într-o variabilă.
- La fiecare iterație, variabilă se va suprascrie cu valoarea actuală.
- Rând pe rând, se vor parcurge toate elementele dintr-o colecție.

```
culori = ['rosu', 'albastru', 'verde', 'galben', 'mov']
for culoare in culori:
    print(f'Culoarea mea preferata e {culoare}')
```

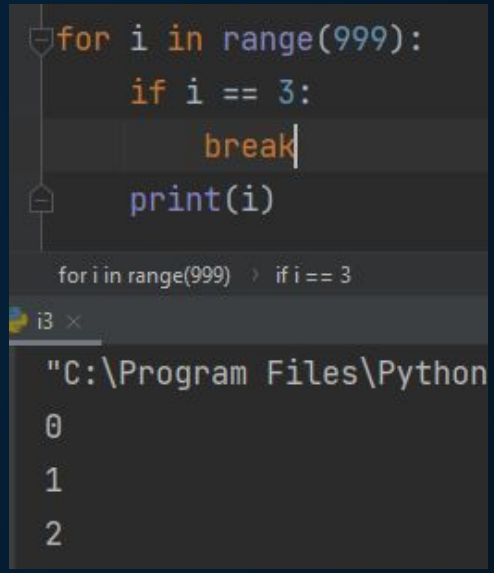
for culoare in culori

i3

```
"C:\Program Files\Python310\python.exe" "C:/IT Factory/Pr
Culoarea mea preferata e rosu
Culoarea mea preferata e albastru
Culoarea mea preferata e verde
Culoarea mea preferata e galben
Culoarea mea preferata e mov
```

Break

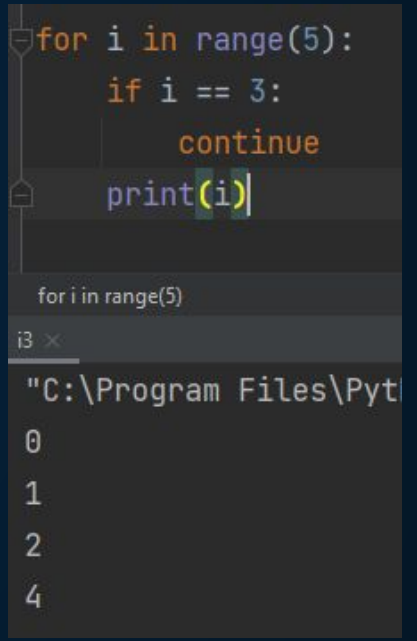
- Cuvântul cheie ‘break’ va opri iterația.
- Practic se iese automat din loop.
- Nu se mai execută codul de după break, din cadrul unui for/ while.



{iT}Factory

Continue

- Cuvântul cheie ‘continue’ va sări peste iterația actuală.
- E un fel de skip.
- Se va sări peste blocul de cod de după skip, care ține de for/ while.



{iT}Factory

Întrebări de interviu:

- Când folosim break?
- Care este diferența dintre For și While?
- Când folosim Continue?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 5

Funcții

Obiective Întâlnire 5

- Să învățăm să lucrăm cu funcții
  - Funcții simple
  - Funcții cu parametri
  - Funcții cu return
  - Funcții cu parametri și return

Ce este o funcție?

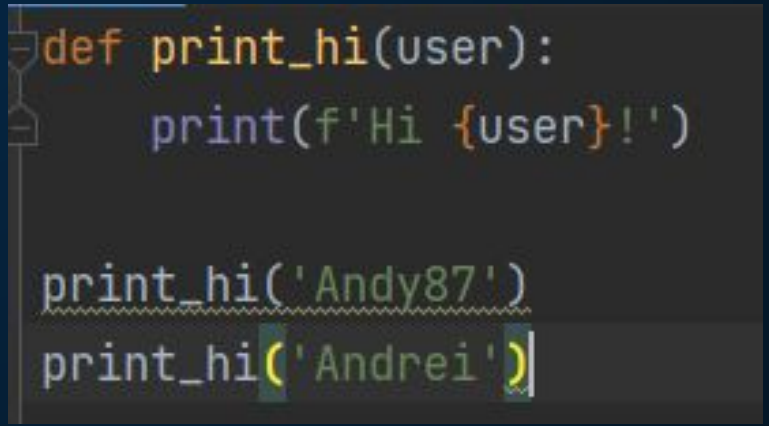
- O zonă de cod cu o mică logică proprie, care poate fi folosită/refolosită (apelată) de oricâte ori avem nevoie.
- Asta e și utilitatea ei principală: ajută să eliminăm copy paste
- Write once, use n times

```
# un fel de program files
# unde sunt instalate programele
def print_hi():
    print('Hi!')

# un fel de desktop
# unde avem shortcut catre ele
print_hi()
print_hi()
```

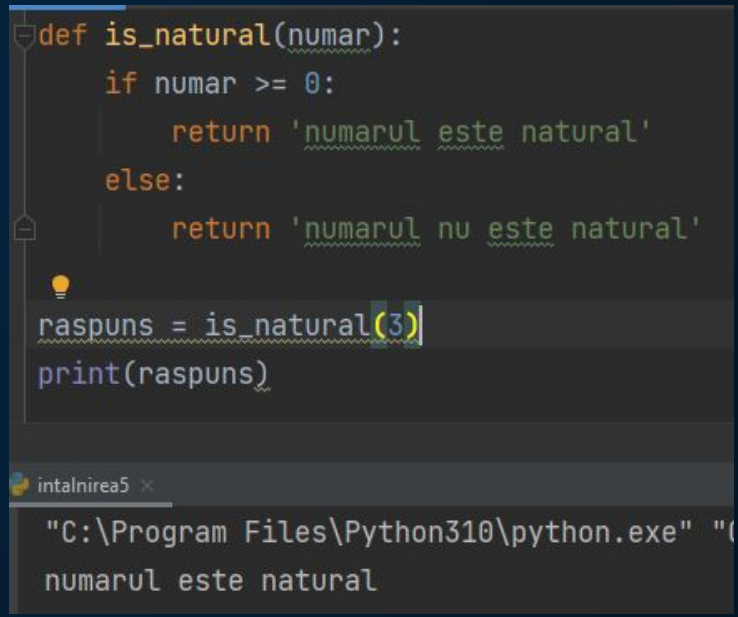
Ce este un parametru?

- Datele de intrare (input) într-o funcție
- Uneori funcția are nevoie de niște date că să poată funcționa
- De exemplu, dacă ar fi să continuăm funcția print_hi și să o customizam pentru diferiți utilizatori vom avea nevoie de un parametru unde să păstrăm utilizatorii diferiți
- O funcție poate să aibă oricâți parametri
- Params sunt opționali
- Dacă avem mai mulți, se despart de
- Practic sunt niște variabile declarate dar neinițializate
- Ele vor fi inițializate (adică vor primi valori), la apelarea funcției



Ce este un return?

- Se folosește când funcția ne și expune un răspuns (output)
- Acest răspuns se poate salva în variabile
- Return e opțional
- Se poate returna orice tip de date cunoscut
- În general, return e ultimul lucru în funcție, deoarece aici se iese din functie
- It's like a lovechild between break + pop :)) (iese din funcție dar ne oferă și un răspuns)
- În general avem un singur return. Excepție când folosim if else, atunci putem avea mai multe, dar oricum la rulare se va ajunge doar într-un singur caz



{iT}Factory

Întrebări de interviu:

- Ce este o funcție?
- Ce este un parametru?
- Ce este un return și când este folosit?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 6

OOP

Obiective Întâlnire 6

- Să înțelegem programarea bazată pe obiecte
- Ce este o clasă?
- Ce este un obiect?
- Ce sunt atributele/field-urile?
- Ce sunt metodele?
- Ce este un constructor?
- Cum importăm clase din alte fișiere?

Ce este o clasă?

- O clasă este o rețetă (blueprint) pentru crearea obiectelor

- Ea conține elemente descriptive: atribute/fields (practic niste variabile)
- Conține acțiuni posibile: metode (practic niște funcții)
- Self - este instanță clasei, ajută funcția să aibă acces la atributele clasei

- Deci o clasă este doar un concept, cum ar fi o rețetă pentru paste carbonara. Faptul că există rețetă nu înseamnă că există și pastele.

- Dar aceeași rețetă o putem folosi ca să facem 1, 2, 100 de porții carbonara.

Ce este un obiect?

- Obiect = instanță a clasei
- Toate obiectele de tip Car vor avea același comportament
  - Aceleași atribute
  - Aceleași metode
  - Atributele pot suferi modificări după inițializarea obiectului
  - Ex: o mașină se poate vopsi într-o culoare nouă

- Putem crea oricâte obiecte de tip Car dorim
- Acesta e și avantajul OOP: write once, use n times

```
car1 = Car() # initializam obiecte de tip Car
car2 = Car() # car2 este instanta a clasei Car
print(car1.make) # dupa . avem acces la fields
print(car2.make)
car1.model = 'Duster' # putem suprascrie valori
car2.model = 'Logan'
car1.accelerate() # dupa . avem acces la methods
car2.accelerate()
car1.stop()
car2.stop()
```

Ce este un constructor?

- Constructor se asigură că la crearea obiectelor setăm niște date fără de care obiectul nu are sens să existe
- Practic atribuie valori atributelor

- Dacă ne gândim la un formular, ar fi acele field-uri cu * care sunt obligatorii

- Dacă prin constructor suntem obligați să dăm model și color, nu am putea să instanțiem obiecte de tip Car fără să dăm aceste valori obligatorii

```
class Car:
    # fields (atribute)
    make = 'Dacia'
    model = None
    year = 2022
    color = None

    # constructor
    def __init__(self, model, color):
        self.model = model
        self.color = color


car1 = Car('Duster', 'white')
car2 = Car('Logan', 'blue')
print(car1.make)
print(car1.model)
print(car1.color)
```

Cum importăm clase din alte fișiere?

- **from folder.folder.fișier import nume_clasă**

```
from folder1.car import Car

car1 = Car('Duster', 'orange')
```

{iT}Factory

Întrebări de interviu:

- Ce este un constructor?
- Care este diferența dintre o clasă și un obiect?
- La ce ne ajută să importăm clase din alte fișiere?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 7

OOP

Obiective Întâlnire 7

- Să știm ce e o excepție și cum o ‘tratăm’
- Să înțelegem cei 4 piloni ai OOP
  - Encapsulare
  - Moștenire
  - Abstractizare
  - Polimorfism

Excepții

- Situații când codul nu poate executa instrucțiunile
- În acest caz codul ‘aruncă’ o excepție
- Programatorii se pot aștepta la ea, pot să o ‘prindă’ și să o ‘trateze’
- În acest sens putem anticipa erori și evităm să ‘crape’ aplicația
- Se folosește sintaxa try/except
- Uneori dorim să ‘ridicăm’ o excepție intenționat

```
try: # in try punem codul 'periculos'
    lista = [1, 2, 3]
    lista[6]
except IndexError as e: #tratam IndexError exception
    print(e)
```

```
# fortezi o exceptie
raise IndexError('Limita elevilor din clasa este 30')
```

Inheritance

- O clasă părinte poate fi moștenită de oricâte clase copil
- Aceste clase copil vor avea acces la toate atributele și metodele clasei părinte

```
class Chef: # clasa parinte
    def make_salad(self):
        print("salad")
    def make_fries(self):
        print("fries")

#clasa copil care mosteneste clasa Chef (se trece intre paranteze numele clasei parinte)
class JapaneseChef(Chef):
    def make_sushi(self):
        print("sushi")

class ItalianChef(Chef):
    def make_pizza(self):
        print("pizza")
```

Polymorphism

- **Când există 2 funcții cu același nume dar au comportament diferit**

```python
# Polymorphism with class methods
class Romania():
    def language(self):
        print("Romanian")
class USA():
    def language(self):
        print("English")
obj_ro = Romania()
obj_usa = USA()
for country in (obj_ro, obj_usa):
    country.language()
```

```python
# user polymorphic function
def add(x, y, z=0):
    return x + y + z

print(add(2, 3))
print(add(2, 3, 4))
```

```python
# ex de built in polymorphic function
print(len("abc"))
print(len([1, 2, 3, 4]))
```

Abstraction

- O clasă abstractă conține metode fără corp dar și metode cu logică definită
- O Interfață conține doar metode abstracte
- Aceste clase vor fi moștenite de clasele copil care vor trebui să scrie logica metodelor
- “Dog() class implements the Animal() Inteface”
- Clasa părinte e că o rețetă ce trebuie implementată exact așa de către toți moștenitorii

```python
class Animal(ABC):

    @abstractmethod
    def sound(self):
        raise NotImplementedError

    @abstractmethod
    def sleep(self):
        raise NotImplementedError
```

```python
class Dog(Animal):
    def sound(self):
        print('Ham Ham!')

    def sleep(self):
        print('ZZZZZZZ')
```

```python
class Cat(Animal):
    def sound(self):
        print('Miau Miau!')

    def sleep(self):
        print('prrrrr')
```

Encapsulation

- În general, că să nu aglomerăm opțiunile utilizatorului, atributele se ascund
- În loc să vadă toate fields și methods va vedea doar metodele
- Păstrăm codul clean/curat
- Și metodele care nu se doresc a fi expuse pot fi ascunse
- Se folosește sintaxa __fieldName sau __methodName

```
class Car:
    __color = 'gray'

    def get_color(self): # folosim getter sa afisam culoarea
        return self.__color

    def set_color(self, culoarea_dorita): # folosim setter ca sa setam o alta culoare
        self.__color = culoarea_dorita

    def __hidden(self):
        pass
```

Encapsulation (optional)

- Decoratorul @property ne ajută să folosim getter, setter, deleter într-un mod ‘Pythonic’

```python
class CarPy:

    def __init__(self, color):
        self.__color = color

    @property
    def color(self):
        return self.__color
```

```python
    @color.getter
    def color(self):
        print(f'Getter: Culoarea este {self.__color}')
        return self.__color

    @color.setter
    def color(self, color):
        print(f'Setter: Noua culoare este {color}')
        self.__color = color

    @color.deleter
    def color(self):
        print(f'Deleter: Am sters culoarea')
        self.__color = None
```

```python
car2 = CarPy('gray')
car2.color = 'red' # set color
car2.color # get color
del car2.color # del color
car2.color # get color
```

```
encapsulation
"C:\Program Files\Python310\pyth
Setter: Noua culoare este red
Getter: Culoarea este red
Deleter: Am sters culoarea
Getter: Culoarea este None
```

{iT}Factory

Întrebări de interviu:

- Ce sunt excepțiile și când apar ?
- Ce este Polymorphism-ul ?
- La ce ne ajuta Encapsulation ?
- Ce este Abstraction ?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 8

Selectors

Obiective Întâlnire 8

- Să verificăm dacă are toată lumea proiectul pe github
- Să aprofundăm și să sedimentam diferitele tipuri de selectori
  - Id
  - Link Text
  - Partial Link Text
  - Name
  - Tag
  - Class name
  - CSS
  - Xpath

Id

- În Terminal: pip install webdriver-manager si pip install selenium
- Deschidem Chrome
- Navigăm către url dorit (ex: https://formy-project.herokuapp.com/form)
- Click dreapta pe elementul ce dorim să îl inspectăm
- Selectați opțiunea 'Inspect'
- Veți găsi partea de html a unui website
- Structura e următoarea:
  - tag sau webelement (ex: <input>)
  - atribut="valoare" (ex: type="text" id="first-name")
- Copiem ID al elementului dorit (ex: 'first-name')

```
# selector by ID
first_name = chrome.find_element(By.ID, 'first-name')
first_name.send_keys('Andy')
```

Link Text / Partial link text

```
# navigam catre un url
chrome.get('https://formy-project.herokuapp.com/')

# selector by Link Text
chrome.find_element(By.LINK_TEXT, 'Autocomplete').click()
```

```
# selector by Partial Link Text
chrome.find_element(By.PARTIAL_LINK_TEXT, 'Enabled').click()
```

Name / Tag

```
chrome.get('http://www.seleniumframework.com/Practiceform/')

# selector by Name
chrome.find_element(By.NAME, 'country').send_keys('Romania')
```

```
chrome.get('https://formy-project.herokuapp.com/form')

# selector by Tag - ia primul tot tp. - e ok doar daca avem tag unic
chrome.find_element(By.TAG_NAME, 'input').send_keys('Andy')

# gasim mai multe si le punem in lista
input_list = chrome.find_elements(By.TAG_NAME, 'input')
input_list[1].send_keys('Sinpetrean')
```

Class name

```
chrome.get('https://formy-project.herokuapp.com/form')

# selector by Class - ia primul tot tp. - e ok doar daca avem clasa unica
chrome.find_element(By.CLASS_NAME, 'form-control').send_keys('Andy')

# gasim mai multe si punem in lista
chrome.find_elements(By.CLASS_NAME, 'form-control')[1].send_keys('Sinpetrean')
```

CSS

```
chrome.get('https://formy-project.herokuapp.com/form')

# selector by CSS - ID
chrome.find_element(By.CSS_SELECTOR, 'input#first-name').send_keys('An')

# selector by CSS - Class - only first one if multiple matches
chrome.find_element(By.CSS_SELECTOR, 'input.form-control').send_keys('dy')

# selector by CSS - atribut=valoare
chrome.find_element(By.CSS_SELECTOR, 'input[placeholder="Enter last name"]').send_keys('S')

# selector by CSS - atribut=valoare partiala + parinte -> copil
chrome.find_element(By.CSS_SELECTOR, 'div input[placeholder*="last name"]').send_keys('inpetrean')
```

XPATH

```
chrome.get('https://formy-project.herokuapp.com/form')

# selector by Xpath - atribut=valoare
chrome.find_element(By.XPATH, '//input[@id="first-name"]').send_keys('A')

# selector by Xpath - * toate elementele care resecta regula
chrome.find_element(By.XPATH, '//*[@id="first-name"]').send_keys('n')

# selector by Xpath - navigare in jos - trecem prin fiecare element
chrome.find_element(By.XPATH, '//div/div/input[@id="first-name"]').send_keys('d')

# selector by Xpath - navigare in jos - skip tags - cautam oriunde sub form un input care sa respecte regula
chrome.find_element(By.XPATH, '//form//input[@id="first-name"]').send_keys('y')

# selector by Xpath - selectare elem din lista - index incepe de la 1
chrome.find_element(By.XPATH, '(//input[@class="form-control"])[2]').send_keys('S')
```

XPATH

```
# selector by Xpath - OR primul gasit dintre variante
s = chrome.find_element(By.XPATH, '//input[@id="last-name1"] | //input[@id="last-name2"] | //input[@id="last-name"]')
# stergem valorile din input
s.clear()
s.send_keys('Sinpetrean')

# selector by Xpath - in f de textul partial + luam textul de pe el cu proprietatea text
full_text = chrome.find_element(By.XPATH, '//a[contains(text(), "Sub")]').text
print(full_text)
```

```
# selector by Xpath - in f de textul vizibil
chrome.find_element(By.XPATH, '//a[text()="Submit"]').click()
```

XPATH - axis navigation

```
'''
x y axis navigation
cu parent in sus
cu /elem_type - ajungem la elementul copil
cu //elem_type - cauta prin toti descendentii
cu parent::elem_type in sus
cu following-sibling::elem_type - urmatorul frate de la acelasi nivel
cu preceding-sibling::elem_type - fratele anterior de la acelasi nivel
//label[text()="First name"]/parent::strong/following-sibling::input/preceding-sibling::strong
'''
```

XPATH - axis navigation

```python
# cu ajutorul unei functii cand avem foarte multe elemente de acelasi tip
# si vrem sa parametrizam selectorul
def formy_input(placeholder_text, input_value):
    input = chrome.find_element(By.XPATH, f'//input[@placeholder="{placeholder_text}"]')
    input.clear()
    input.send_keys(input_value)

formy_input('Enter first name', 'ANDY')
formy_input('Enter last name', 'SINPETREAN')
formy_input('Enter your job title', 'QA AUTOMATION ENGINEER')
```

{iT}Factory

Întrebări de interviu:

- Câte tipuri de selectors/locators cunoașteți?
- Care considerați că e de preferat să se folosească? De ce?
- Care e cel mai flexibil din punctul tău de vedere? De ce?
- Exercițiu practic: Se va da un url și câteva elemente pe care să le identificați voi după ce selector doriți.

Demo sites for practice:
https://www.phptravels.net/
http://automationpractice.com/index.php
https://formy-project.herokuapp.com/
https://the-internet.herokuapp.com/
https://www.techlistic.com/p/selenium-practice-form.html

Întrebări & curiozități?

{iT}Factory

Întâlnirea 9

Verificări

Obiective Întâlnire 9

- Să știm să așteptăm elementele corect
  - Implicit wait
  - Explicit wait
- Să știm să facem verificările necesare la finalul unui test folosind
  - Cuvântul rezervat ‘assert’
  - Librăria unittest
  - Simple tests în pytest
  - Opțional - rulare în paralel cu addon pytest
    - pytest parallel
    - pytest xdist

Waits

- Uneori robotelul vrea să interacționeze cu elementele înainte ca acestea să devină active
- În acest caz avem 3 variante

1. sleep() - oprește execuția codului pentru un nr de sec
2. implicit wait - stabilește câte secunde să caute selenium după orice element
3. explicit wait - stabilește câte secunde să caute selenium după un element particular

```
# pauzam codul x secunde
sleep(3)
```

```
# setam implicit wait in secunde
# selenium va cauta toate elementele timp de x secunde inainte sa dea eroare
chrome.implicitly_wait(5)
```

```
# cauta elementul timp de 10 secunde (refresh la fiecare 500ms)
last_name = WebDriverWait(chrome, 10).until(EC.presence_of_element_located((By.ID, "last-name123")))
last_name.send_keys('S')
```

Verificări cu assert

```
# navigam catre un url
chrome.get('https://formy-project.herokuapp.com/form')

actual = chrome.current_url
expected = 'https://formy-project.herokuapp.com/form2'

assert actual == expected, f'INVALID URL: expected {expected} but found {actual}'

print('TEST PASSED')
```

Verificări cu unittest

- Se importă librăria unittest
- Se moștenește clasa TestCase
- În funcția setUp() punem tot ce ne dorim să se execute înainte de teste
- În funcția tearDwon() punem tot ce ne dorim să se execute după teste
- Scriem câte metode de test avem nevoie
- În mod ideal fiecare test case conține o singură verificare
- Putem rula test case individuale - fiecare trebuie să fie independent și funcțional
- Putem rula toată clasa de teste
- Data viitoare vom învăța cum să le punem în suite de teste (să le grupam logic)

{iT}Factory

Verificari cu unittest

```python
class Test(unittest.TestCase):

    # se rulaza inainte de fiecare test
    def setUp(self):
        s = Service(ChromeDriverManager().install())
        self.chrome = webdriver.Chrome(service=s)
        self.chrome.maximize_window()
        self.chrome.get('https://formy-project.herokuapp.com/form')

    # se ruleaza dupa fiecare test
    def tearDown(self):
        self.chrome.quit()

    # verificam URL
    def test_url(self):
        actual = self.chrome.current_url
        expected = 'https://formy-project.herokuapp.com/form'
        # expected value, actual value, mesaj in caz de fail
        self.assertEqual(expected, actual, 'URL is incorrect')
```

{iT}Factory

Întrebări de interviu:

- Ce este un constructor?
- Care este diferența dintre o clasă și un obiect?
- La ce ne ajută să importăm clase din alte fișiere?
- De ce nu e recomandat sa folosim sleep în cod?
- Care e diferența dintre implicit wait și explicit wait?
- Dacă avem și implicit și explicit wait care e folosit?
- Ce librarie se folosește pentru a face verificări la final de test?
- Ce metode se folosesc pentru a face verificări la final de test?
- Dacă 'pică' testul la o verificare, mai continuă sau se oprește?
- Putem face verificări la început de test?
- Cum verificăm dacă un element nu e vizibil? (2 modalități)
- Ce face funcția setUp()?
- Ce face funcția tearDown()?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 10

Extra

Obiective Întâlnire 10

- Să știm să unim mai multe teste în suite
- Să știm să folosim și alte browsere
- Să realizăm primele rapoarte html
- Să reușim să rezolvăm probleme extra cum ar fi:
  - Alerte
  - Autentificări
  - Context menu
  - Să apăsăm taste
- Paralel run*

Suite de teste + rapoarte + skip test

- Suitele ne permit să rulăm mai multe teste deodată
- pip install html-testRunner
- În metoda addTests adaugăm o listă cu toate clasele de test dorite
- Pentru skip folosim decoratorul @unittest.skip

```
import unittest
import HtmlTestRunner # pip install html-testRunner

from ta8.sesiuni.intalnirea9.test4_unittest import Test
from ta8.sesiuni.intalnirea9.test5_ex import Test2


class TestSuite(unittest.TestCase):

    def test_suite(self):
        smoke_test = unittest.TestSuite()
        smoke_test.addTests([
            unittest.defaultTestLoader.loadTestsFromTestCase(Test),
            unittest.defaultTestLoader.loadTestsFromTestCase(Test2),
        ])

        runner = HtmlTestRunner.HTMLTestRunner(
            combine_reports=True,
            report_title='Test',
            report_name='Smoke Test Result'
        )

        runner.run(smoke_test)
```

Alte browsere

- https://pypi.org/project/webdriver-manager/

Situații extra

- Alert are doar ok
- Confirm are ok / cancel
- Prompt are input + ok / cancel
- Sintaxa corectă când ne autentificăm într-o pagină:
  - https://user:pass@the-internet.herokuapp.com/basic_auth
- Context menu: meniu care se deschide cu click dreapta și e mai greu de simulat

```
action = ActionChains(self.chrome)
action.context_click(self.chrome.find_element(*self.BOX)).perform()
```

- Apăsare taste de la tastatură

```
user.send_keys(Keys.CONTROL + 'a')
sleep(1)
user.send_keys(Keys.BACKSPACE)
```

{iT}Factory

Întrebări de interviu:

- Diferența dintre alert, confirm și prompt?
- Ce este o suită de teste și la ce ne ajută?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 11

BDD, POM

Obiective Întâlnire 11

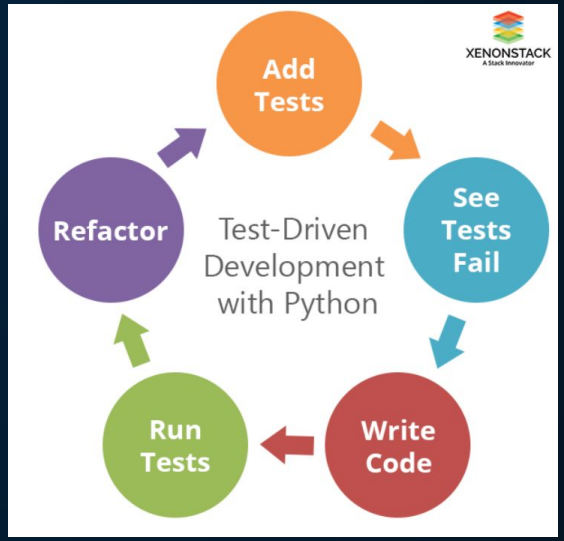
- Să începem proiectul final de web testing
- Să înțelegem metodologia behavior driven development (BDD)
- Să înțelegem ce sunt termenii de epic/user story/acceptance criteria
- Să înțelegem page object model (POM) design pattern
- Să știm să generăm singuri oricâte pagini avem nevoie într-un proiect

Final BDD Project

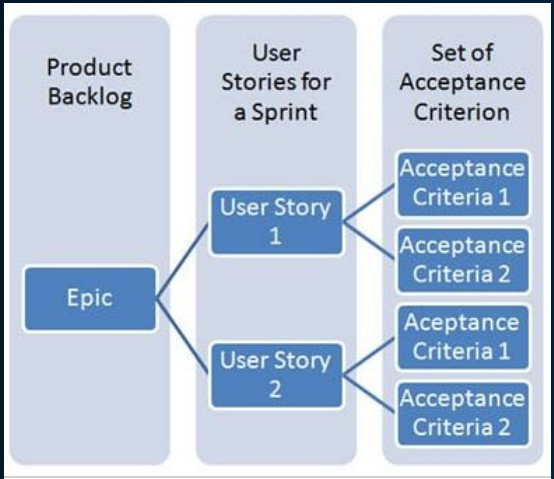
- Pycharm -> New Project
- pip install selenium (sau pip install -U selenium - pt update la zi)
- pip install behave (o librărie bdd care vă interpreta și rula testele scrise în gherkin)
- pip install behave-html-formatter (ne ajută să generăm rapoarte bdd)
- pip install webdriver-manager
- În fișierul browser initializam driver
- Creăm un folder ‘pages’ în care vom face clasele pentru fiecare pagină din site

BDD

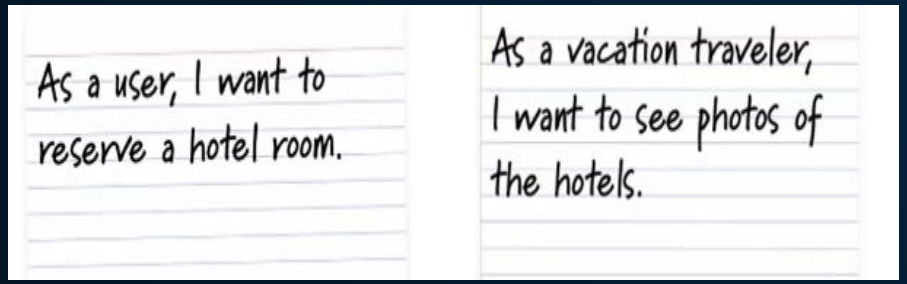
- Este o metodologie derivată din TDD (test driven development)
- Focus mare pe testare. Se începe cu scrierea testelor. Se rulează testele care vor pica. Se scrie codul aplicației. Se rulează testele din nou și care vor trece acum.
- În bdd scriem testele în plain English, cu ajutorul sintaxei gherkin.
- Avantaj: Toate persoanele interesate (Clienți, Manageri, BAs, Developeri, Testeri etc) vor înțelege ușor rapoartele generate. Acestea reprezintă 'living documentation' pentru Proiect.
- Avantaj: Focusul se mută pe calitatea produsului și pe testare. Știm sigur că avem timp de teste și că acestea nu se vor ignora deoarece avem un 'test first approach'.
- De cele mai multe ori un product owner generează user stories care conțin acceptance criteria sub forma given, when, then. Noi trebuie doar să le automatizam 1:1 -> easy life.

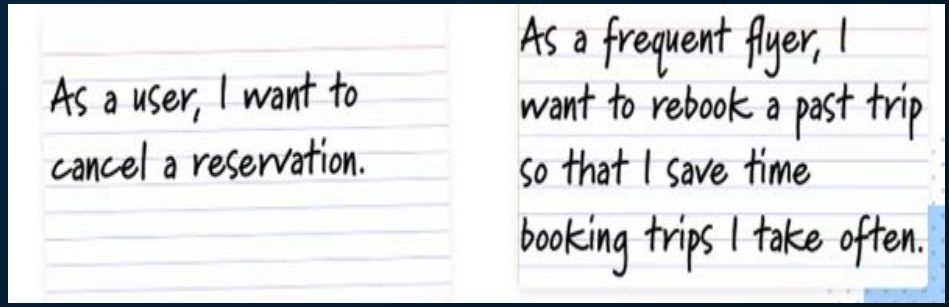


Agile, Sprint, Epic, User story, Acceptance criteria

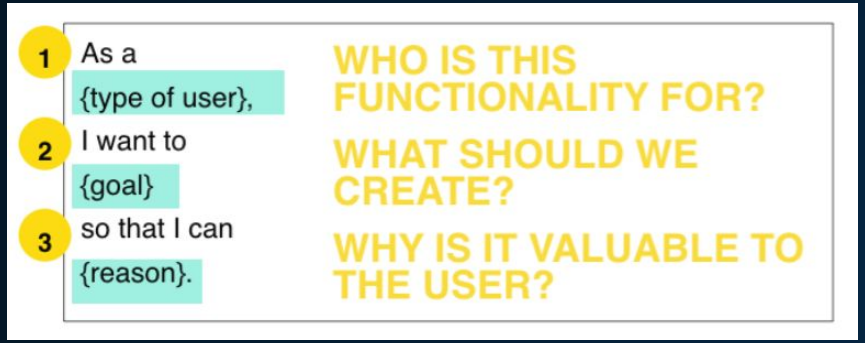


- **Agile: metodologie în care proiectul se împarte în bucăți funcționale. Focus pe colaborare, viteza de lucru, delivery și improvement constant prin feedback loop.**
- **Sprint: o perioadă de 1-2 sapt. În care se livrează o bucată funcțională din proiect.**
- **Epic: o funcționalitate majoră (de ex: register user).**
- **Ușer story: o explicație end to end a unui feature făcută din perspectiva utilizatorului final.**
- **Acceptance criteria (DOD - definition of done): cerințe predefinite care trebuie îndeplinite pentru a închide un story.**





Examples



As a registered user, I want to log in with my username and password so that the system can authenticate me and I can trust it.

1. Given that I am a registered user and logged out, if I go to the log in page and enter my username and password and click on Log in, then the data associated to my user should be accessible.

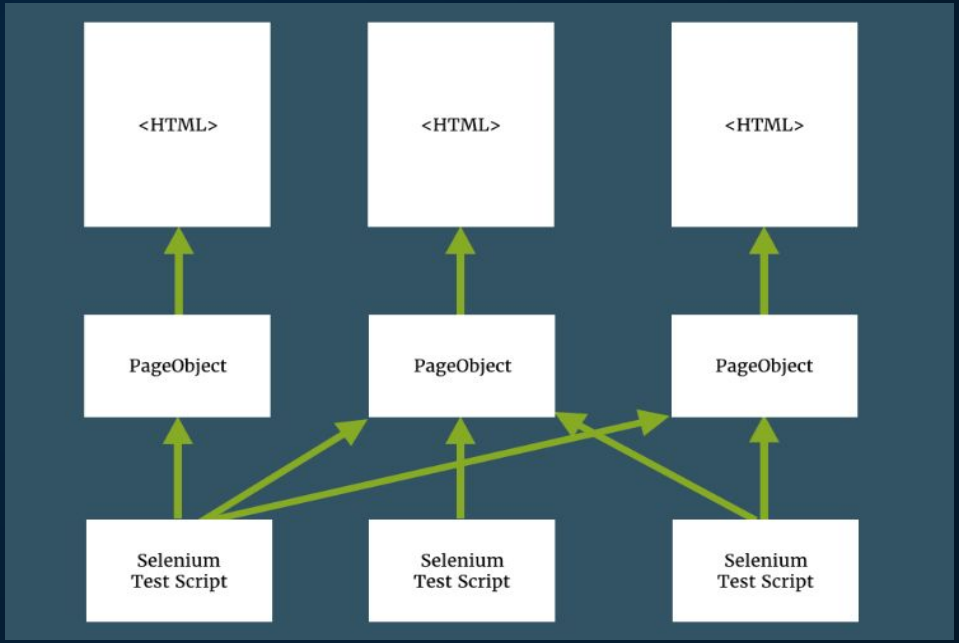
2. Given that I am a registered user and logged out, if I go to the log in page and enter my username but an incorrect password and click on Log in, then log in fails with an error message that specifies that the username or password was wrong.

As a new user, I want to register by creating a username and password so that the system can remember me and my data.

Given that I am a new user, when I go to the sign up page and enter an username and password and click on sign up, then I am successfully registered and able to log in with my chosen credentials.

POM

- Page object model este cel mai folosit design pattern în testarea web
- Ne ajută să organizăm fișierele într-un mod logic și ușor de folosit
- Fiecare pagină din proiect va avea o clasă atașată unde găsim elementele din ea și interacționăm cu acestea.



Gherkin syntax

- Metoda mai ușoară și general acceptată de a scrie test cases
- Folosită mai ales în metodologia BDD (behavior driven development)
- Are 5 cuvinte cheie: Given, When, Then, And, But
- Cu ajutorul lor descriem comportamentul unui utilizator în aplicația testată

Feature File:

```
1  Feature: login
2  Scenario: login with valid credentials
3    Given I am on the login page
4    When I enter a valid email
5    And I enter a valid password
6    And I press "Login"
7    Then I should be on the users home page
8    And I should see "Login successful"
9
10 Scenario: login with invalid username
11   Given I am on the login page
12   When I enter a invalid email
13   And I enter a valid password
14   And I press "Login"
15   Then I should see an error message
16
17 Scenario: login with invalid password
18   Given I am on the login page
19   When I enter a valid email
20   And I enter a invalid password
21   And I press "Login"
22   Then I should see an error message
```

{iT}Factory

Întrebări de interviu:

- Dacă ar trebui să începi un framework de la 0, de ce ai alege BDD?
- Cunoști POM design pattern? Care ar fi avantajul folosirii lui?
- Ce e un epic? User story? Acceptance criteria?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 12

Final Project

Obiective Întâlnire 12

- Să finalizăm proiectul final
- Să facem un test cap coadă, finalizat cu assert și raport html
- Să știm să parametrizam testele și să folosim tabele de valori
- Să înțelegem structura proiectului astfel încât să putem scrie singuri teste

Steps Folder

- Conține 1:1 cate o pagină de pași pentru fiecare pagină din proiect
- Deasupra metodelor scriem scenariile cu sintaxa gherkin folosindu-ne de decoratorii aferenți (@given, @when, @then, @and, @but)

```python
@given('login: user is on the login page')
def step_impl(context):
    context.login_page.navigate_to_jules()

# username password parameters are set in login.feature
@when('login: user enters valid username "{username}" and valid password "{password}"')
def step_impl(context, username, password):
    context.login_page.login(username, password)

@when('login: user clicks on forgot pass button')
def step_impl(context):
    context.login_page.click_forgot_pass_btn()

@then('login: verify that invalid credentials error is displayed')
def step_impl(context):
    context.login_page.verify_error_displayed()
```

{iT}Factory

Features folder

- Conține fișiere de tip .feature în care scriem scenariile de test

```
Feature: Jules login feature

    # se ruleaza inainte de fiecare test
    Background:
      Given login: user is on the login page

    @smoke
    Scenario: Login fara tabel de valori
      When login: user enters valid username "itfactory.ro@gmail.com" and valid password "pass123"
      Then login: verify that invalid credentials error is displayed


    @test
    Scenario Outline: Login cu tabel de valori
      When login: user enters valid username "<email>" and valid password "<pass>"
      Then login: verify that invalid credentials error is displayed

    Examples:
      | email                  | pass      |
      | itfactory1.ro@gmail.com | password1 |
      | itfactory2.ro@gmail.com | password2 |
      | itfactory3.ro@gmail.com | password3 |
```

Features folder

- Aici ținem testele. Fiecare feature file testează o funcționalitate majoră a aplicației și poate conține mai multe scenarii de tests
- În Feature trecem titlul capabilității testate
- În Background trecem pașii care se execută înainte de fiecare test
- În Scenario: punem titlul testului
- Sub Scenario trecem pașii din test cu sintaxa gherkin
- Scenario Outline: ne permite să folosim un tabel de valori
- @smoke și @test sunt taguri care au rolul să împartă testele în categorii - aceste nume de taguri sunt la liberă alegere de către voi
- În Examples punem parametri pentru teste într-o structură tabelară
- Fiecare rând din tabel va fi un test run individual

{iT}Factory

Întrebări de interviu:

- De ce ar fi bine să avem o clasă BasePage în proiect?
- Ce conține o clasă de tip page?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 13

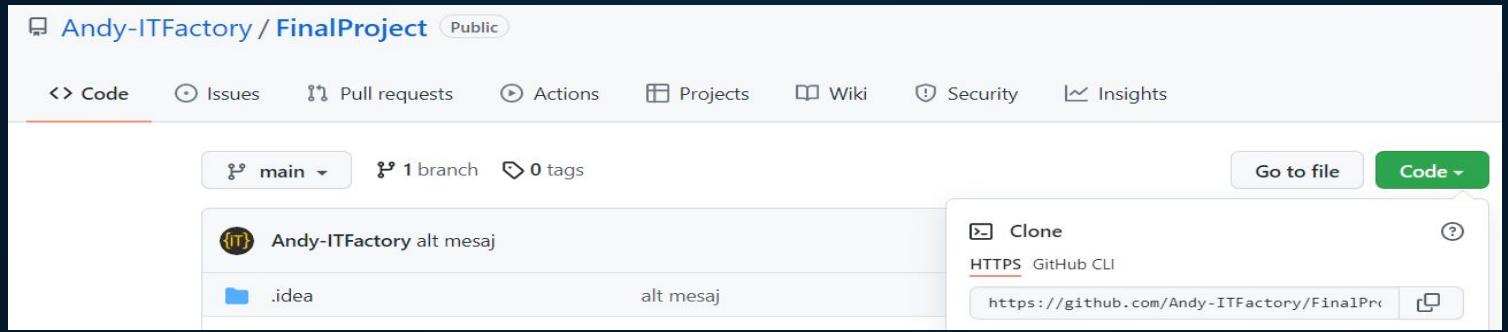
Final Project v2

Obiective Întâlnire 13

- Să clonăm un proiect din git (adică să importăm din cloud la noi pe PC)
- Să instalăm dependințele necesare
- Să înțelegem cum se leagă toate cap coadă
- Să putem să rulăm teste și să generăm rapoarte
- Să putem să extindem proiectul
- Să știm cum se face debug cu behave

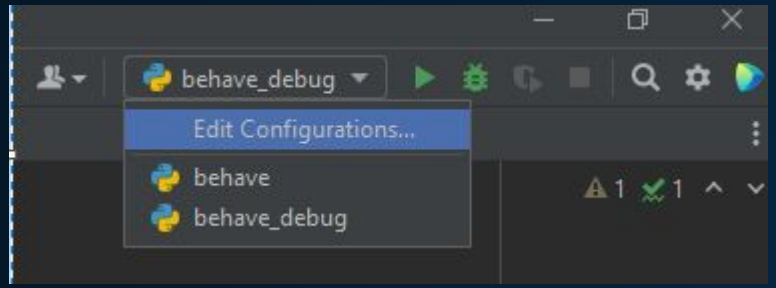
Git clone

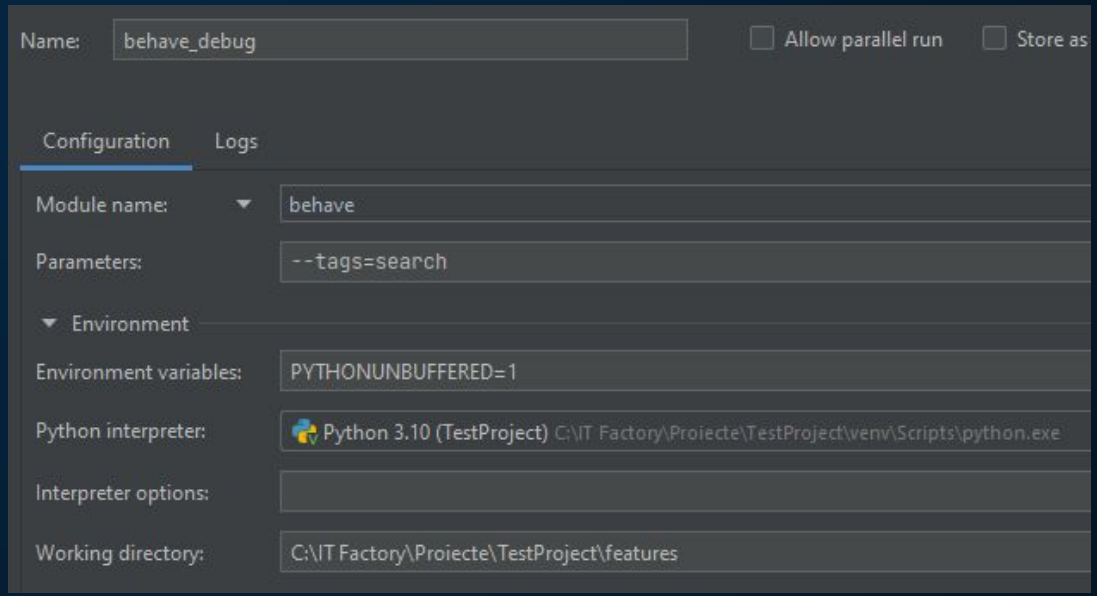
- Cum importăm un cod din github?
  - git clone url_proiect (ur îl luăm ca în poza de jos)
    - Ex: git clone https://github.com/Andy-ITFactory/emag
  - Dacă nu noi suntem ownerii proiectului ștergem .git și urmăm de la 0 pașii pentru a crea un proiect nou în git în contul nostru (atenție: show hidden files să vedeți .git)
  - Dacă importăm din contul nostru de git, putem continua direct să editățm și să urcăm fișierele sus așa cum știm deja
    - git add .
    - git commit -m “mesaj”
    - git push



{iT}Factory

How to debug in a BDD project





{iT}Factory

Întrebări de interviu:

- Ce este un feature file?
- Ce face Background keyword în cadrul unui feature file?
- Care e diferența dintre un Scenario și un Scenario Outline?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 14

Unit testing

Obiective Întâlnire 14

- Să știm să scriem unit tests (=> repo separat pentru angajator)
- Să avem Postman instalat
- Să cunoaștem cele mai uzuale HTTP response codes
  - 1xx, 2xx, 3xx, 4xx, 5xx
- Să înțelegem ce e un API și să știm cele mai folosite metode
  - Get, post, patch, put, delete
- Să putem să facem câteva request-uri manual în Postman

{iT} Factory

Ce este un Unit Test?

Testarea unitară reprezintă testarea la cel mai low level, practic este o funcție care testează o altă funcție.
Funcția de test apelează funcția testată și se asigură că ea returnează datele corecte, folosind un assert.
De obicei această testare este făcută de devi, dar și un qa o poate face, depinzând de raportul de forțe dev-qa. În companiile cu mai mulți angajați qa această sarcină poate fi trecută în responsabilitatea lor.

```python
# functia ce trebuie testata
def aria_dreptunghiului(self, l, L):
    return l * L
```

```python
# unit test
def test_aria_dreptunghiului():
    assert aria_dreptunghiului(3, 5) == 15, 'Aria nu se calculeaza corect'
```

{iT}Factory

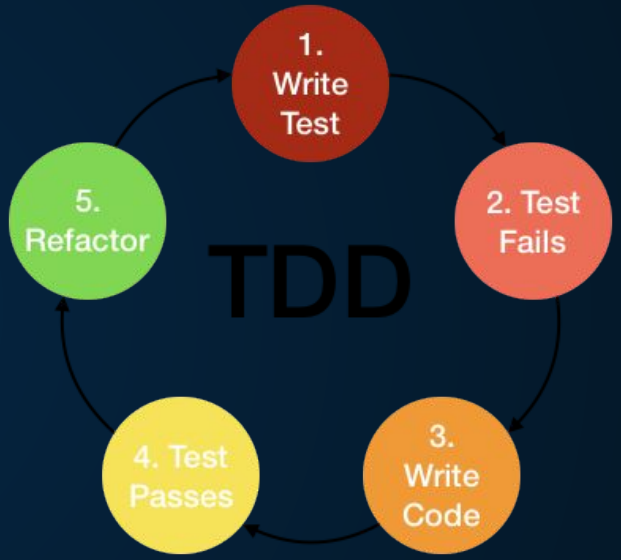
What is TDD?

DD = test driven development
Metodologie care plasează o importanță majoră pe testare.

Se folosește un test first approach.
Tot developmentul începe prin scrierea de teste.

Exercițiu:

1. Instalați pytest (librărie care ne ajuta să rulăm teste)
   pip install pytest
1. Implementați un mini calculator folosind TDD
   a. Clasa MiniCalculator
   b. Atribute: 2 numere a si b
   c. Metode goale de +,-,*,/ folosind ‘pass’
   d. Implementăm testele
   e. Executăm testele (pica)
   f. Implementăm logica
   g. Executăm testele (trec)



Cum se execută testele?
Toate din folder: ‘pytest .\folder_name\’
Un sg. Fisier: ‘pytest .\folder_name\file_name.py’

{iT} Factory

Ce este HTTP/HTTPS?

HTTP/S = Hypertext transfer protocol / secure
Protocol de comunicare între client și server.
Ne ajută să transferăm date prin rețea.

În imagine avem arhitectura standard pe 3 nivele (layers) a unei aplicații.

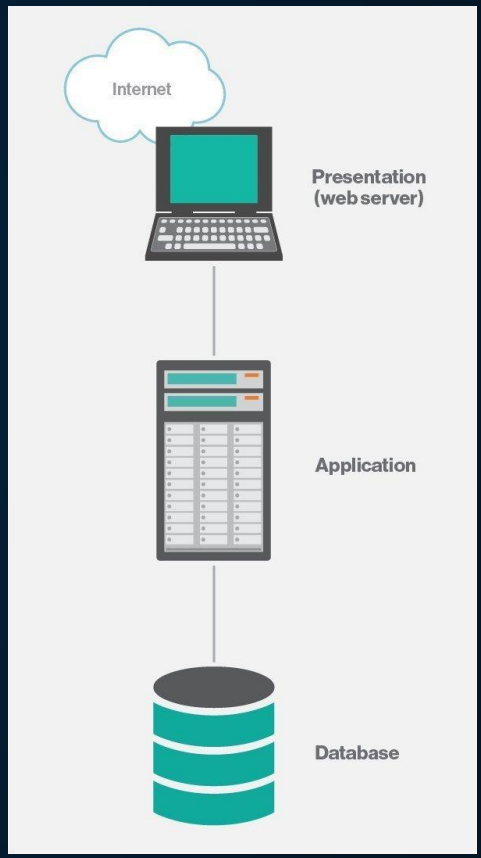
Client <--> HTTPS API requests <--> Server <--> db queries <--> Database

Testing:

UI level (user interface) / Client -> web testing (selenium - știm)

App -> Unit Testing (știm)

API level (comunicarea dintre Client și Server) -> API testing (învățăm)



{iT} Factory

Raspunsuri HTTP

https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

Cele mai folosite:

- 200 OK - succes - de obicei când se cer date de la server
- 201 Created - success - când se pun date în server
- 204 No Content - success - de obicei când ștergem ceva

- 400 Bad request - ceva nu a mers bine, probabil valori invalide pentru parametri
- 401 Unauthorized - nu suntem logați în app
- 403 Forbidden - suntem logați dar nu avem drepturi de edit de exemplu
- 404 Not Found - nu găsește endpoint - probabil
- 408 Request Timeout - a durat prea mult până să ajungă la server requestul

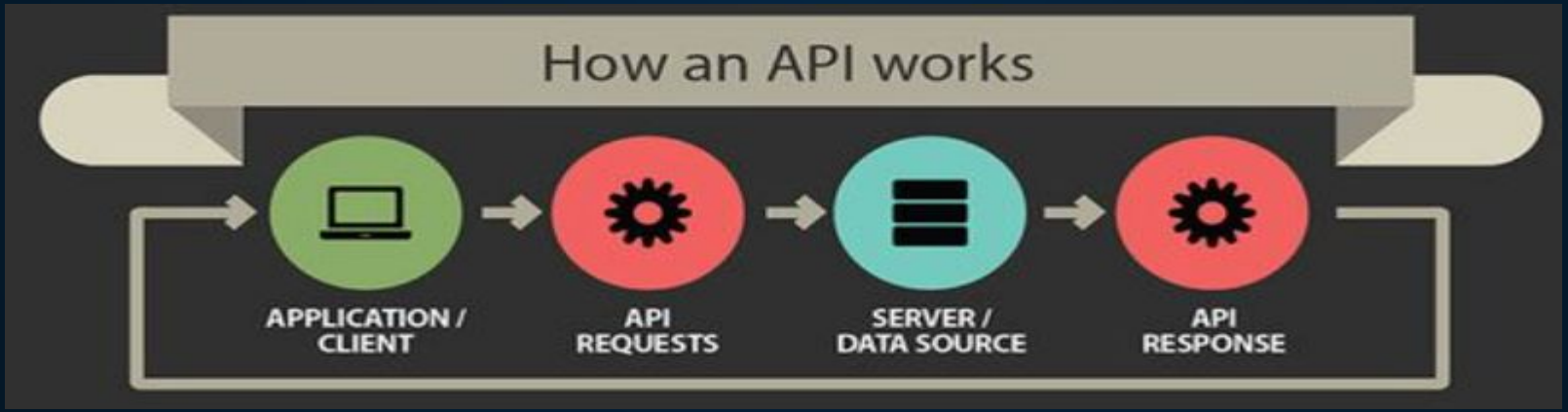
- 500 Internal Server Error - requestul ajunge la server dar cel mai probabil este un bug
- 503 Service Unavailable - serverul e oprit pentru mentenanță de exemplu

Ce este API?

API - application programming interface (‘Interfața de programare a aplicațiilor’)

Niște comenzi bine documentate care ajută programatorul sau interafața aplicației să comunice cu logica din spate.
Clientul trimite un request către un endpoint
Dacă este pornit, serverul ascultă non stop și este pregătit să primească și să proceseze aceste cerințe
După ce serverul procesează informațiile primite/cerute oferă un răspuns clientului
Clientul interpretează răspunsul și îl afișează într-un mod user-friendly utilizatorului (avantaj: un singur API poate fi folosit de orice tip de client - iOS, Android, web etc și de oricâte device-uri)

Schimbul de date se face sub forma unui JSON. Acesta are o structură asemănătoare unui dictionar din Python (cheie:valoare)



{iT}Factory

Metode API

Acronimul CRUD vine de la create, read, update, delete - acțiunile principale când lucrăm cu date în software development

Cele mai folosite metode API sunt:

GET - cerem date de la server - în general 200

POST - trimitem date la server - în general 201

PATCH - updatăm date prin updatarea anumitor atribute ale obiectului (ex: din user doar prenumele) - în general 200 sau 201

PUT - updatăm date prin suprascrierea întregului obiect (ex: tot userul - nume, prenume, adresă) - în general 200 sau 201

DELETE - ștergem date - în general 204

Exerciții:
Le vizualizăm pe toate într-un website folosind developer tools (F12 -> network)
Le implementăm în Postman
Folosim API:
https://documenter.getpostman.com/view/4012288/TzK2bEa8#abe537df-fccc-4ee6-90d2-7513e3024d6b
Implementăm un get, post, patch, put, delete
Studiem JSON-urile într-un parser: http://json.parser.online.fr/

{iT}Factory

Întrebări de interviu:

- De la ce vine TDD? Ce este particular în această metodologie de dezvoltare?
- Ce este un unit test?
- Cu ce cifră încep request-urile HTTP de succes?
- Care e diferența dintre response code 4xx și 5xx?
- De la ce vine API?
- Care sunt metodele principale API? (5)
- Care e diferența dintre patch și put?

Întrebări & curiozități?

{iT}Factory

Întâlnirea 15

API testing

{iT}Factory

Obiective Întâlnire 15

- Recapitulare Postman - CRUD
- API Automation Framework - starter kit

{iT} Factory

Metode API

Acronimul CRUD vine de la create, read, update, delete - acțiunile principale când lucrăm cu date în software development

Cele mai folosite metode API sunt:

GET - cerem date de la server - în general 200

POST - trimitem date la server - in general 201

PATCH - updatăm date prin updatarea anumitor atribute ale obiectului (ex: din user doar prenumele) - 2xx

PUT - updatăm date prin suprascrierea întregului obiect (ex: tot userul - nume, prenume, adresa) - 2xx

DELETE - ștergem date - în general 204

Recap Postman:
https://github.com/vdespa/introduction-to-postman-course/blob/main/simple-books-api.md

{iT}Factory

API Automation

- Instalăm pachetul care ne ajută să facem requesturi
- pip install requests
- Începem împreună testarea pentru Books

https://github.com/vdespa/introduction-to-postman-course/blob/main/simple-books-api.md

- Verificăm să meargă pentru toată lumea callul de status
  - Atât în postman
  - Cât și în cod
- Vă arăt eu codul dar discutăm împreună scenariile
- Vă rămâne la temă să implementați voi ulterior, urmărind video
- Nu vă dau fișierele intenționat ca să mă asigur că scrieți voi codul
  - Oportunitate să înțelegeți și mai bine, linie cu linie
- În final, îl veți urca în github că și repo de API testing

{iT}Factory

Întrebări de interviu:

- Ce librării putem folosi pentru API testing în Python?
- Ce este un token? Unde se pune, în header sau în body?
- Ce este un JSON?
- Care e diferența dintre url params și body params?

Întrebări & curiozități?