

Capitolul 1. Notiuni introductive

	pg
1.1 <u>Introducere in programarea Python</u>	02
1.1.1 Despre computer. Notiuni fundamentale	03
1.1.2 Despre software. Notiuni fundamentale	04
1.1.3 De la limbajul uman la limbajul masina	04
1.1.4 Programarea efectiva – pasii de urmat	05
1.1.5 Python - avantaje si aplicabilitate	06
1.1.6 Instalarea Python	07
1.1.7 Fisiere si executabile Python	10
1.1.8 Codul Python. Utilitare	12
1.2 <u>Primele programe in Python</u>	14
1.2.1 Lucrul cu siruri de caractere. Primii pasi in Python, functia “print”	14
1.2.2 Concatenarea si repetitia sirurilor de caractere	16
1.2.3 Secvente de evadare intr-un sir de caractere	16
1.2.4 Lucrul cu numere	17
1.2.5 Lucrul cu variabile	20
1.2.6 Captarea unui sir de la tastatura	21
1.3 <u>Recapitulare</u>	22
1.3.1 Recapitulare	21

1.1 Introducere in programarea Python

Python este un limbaj de programare creat in anul 1991 de Guido van Rossum. Acesta a fost si ramane inca, un lider al comunitatii de dezvoltatori Python. Exista doua versiuni ale limbajului Python: versiunea 2.x, care are avantajul multor module dezvoltate si versiunea 3.x, care vine din urma si va inlocui treptat versiunea 2.7.x, care nu va mai fi dezvoltata o perioada foarte lunga de timp (estimat 2020).

La momentul (re)scrierii acestui capitol ultimele versiuni disponibile erau 2.7.14 si respectiv 3.6.3. Vom studia versiunea 3.x, cu precadere versiunea 3.5.2., pentru ca **reprezinta viitorul Python**. Totusi, vor fi discutate si anumite aspecte care diferentiaza cele doua versiuni.

Python este un limbaj open source. Oricine este interesat poate gasi pachetul de instalare pentru utilizarea/dezvoltarea Python la adresa:

<https://www.python.org/downloads/>.

Exista mai multe implementari, printre care cele mai importante sunt: CPython, IronPython, Jython, PyPy, etc..

Pentru a intelege filozofia Python este de ajuns sa citim si sa aplicam asa numitul “**Zen of Python**” scris in anul 1999 de Tim Peters - <https://www.python.org/dev/peps/pep-0020/> :

1. Frumos este mai bine decat urat;
2. Explicit este mai bine decat implicit;
3. Simplu este mai bine decat complex;
4. Complex este mai bine decat complicat;
5. Orizontal este mai bine decat imbricat;
6. Rar este mai bine decat dens;
7. Lizibilitatea conteaza;
8. Cazurile speciale nu sunt destul de speciale pentru a nu respecta regulile;
9. Practicul bate puritatea;
10. Erorile nu ar trebui parsate;
11. ... cu exceptia cazului in care facem acest lucru explicit;
12. In cazul ambiguitatilor refuza tentatia de a ghici;
13. Ar trebui sa existe o modalitate clara, explicita – si numai una - de a face un lucru;
14. ... cu toate ca acest lucru nu poate fi evident de la inceput daca nu esti olandez :);
15. Acum este mai bine decat niciodata;
16. ... cu toate acestea, uneori este mai bine niciodata decat chiar acum;
17. In cazul in care punerea in aplicare este greu de explicat, este o idee proasta;

18. In cazul in care punerea in aplicare este usor de explicat, poate fi o idee buna;
19. Namespace-urile sunt ca si claxoanele. Hai sa le facem diferite.

Daca ati accesat Python, nu aveti decat sa tastati instructiunea “**import this**” si veti vedea versiunea originala, in limba engleza.

1.1.1 Despre computer. Notiuni fundamentale

Computerul este un instrument care face diferite lucruri pentru noi. Este o *masina* care stocheaza si manipuleaza informatia cu ajutorul unui *program informatic*. Informatia (intrare) va fi prelucrata si va da nastere la informatie intr-un alt format (iesire).

Programul informatic reprezinta instructiuni detaliate, pas cu pas, care-i transmit computerului ce sa faca.

Practic, fara un program adecvat, computerul este o “cutie” inerta. Computerul nu are inteligenta proprie. Este o masina care asteapta, docila, urmatoarea instructiune. De calitatea instructiunilor (a programului, in speta) depinde rezultatul “muncii” computerului.

Principalele **componente** ale computerului sunt urmatoarele:

- Procesorul (Central Processor Unit – CPU) este cel care prelucreaza informatiile primite, returnand informatia in alta forma;
- Memoria interna (Random Access Memory – RAM) este suportul in care sunt stocate temporar informatiile care vor fi transmise pentru prelucrare catre procesor. Memoria RAM este volatila, depinzand de alimentarea cu energie electrica;
- Memoria externa (HDD, SSD, CD, DVD, etc.) constituie suportul pe care sunt stocate informatiile, atat cele care vor fi preluate de memoria RAM in vederea transmiterii catre procesor, cat si cele obtinute dupa prelucrare. Spre deosebire de memoria RAM, memoria externa nu este volatila, asigurand stocarea informatiilor;
- Periferice de intrare (tastatura, mouse, scanner, etc.) ajuta la preluarea informatiilor de intrare;
- Periferice de iesire (monitor, imprimanta, etc.) ajuta la extragerea informatiilor de iesire.

Functionarea computerului poate fi descrisa simplist astfel: programul este copiat din memoria externa in memoria RAM; de aici, instructiunile cuprinse in program sunt transmise cronologic catre CPU, care le executa.

1.1.2 Despre software. Notiuni fundamentale

Un utilizator de computer se asteapta sa lucreze cu aplicatii prietenoase (software).

Software face masina sa functioneze, ii spune ce urmeaza.

Programarea reprezinta activitatea de creare de software.

Programul este o succesiune de instructiuni stocate, o parte din inteligenta creatorului.

Programele controleaza masina, ii spun ce urmeaza sa faca. Daca schimbam programul masina va face altceva, in conformitate cu noul program.

Programatorul utilizeaza diferite instrumente pentru a crea aplicatii atat pentru diversi utilizatori (raspunzand unor nevoi) cat si pentru propriile nevoi (pentru imbunatatirea propriei activitati de programare, testare, debug, etc.).

Treaba programatorului este sa puna instructiunile in ordinea corecta (inteligenta). Masina face restul (capacitate de procesare).

Programatorul trebuie sa aiba atat o viziune de ansamblu asupra programului, cat si asupra celor mai mici detalii ale acestuia, fiind in masura sa-l modularizeze, sa-l imparta in subprograme.

1.1.3 De la limbajul uman la limbajul masina

Pentru a transmite instructiuni catre computer avem nevoie de un limbaj pe care acesta sa-l inteleaga. De ce nu utilizam *limbajul uman*? Raspunsul e simplu: din cauza ambiguitatii acestuia si a multiplelor posibilitati de interpretare. Pentru a face exact ceea ce ne dorim computerul trebuie sa inteleaga exact, fara dubii, ce are de facut. Acest lucru este posibil prin *limbajul masina*. Pe de alta parte, limbajul masina este foarte dificil si greu de inteles de catre om.

Limbajul de programare este un limbaj intermediar, intre limbajul uman si limbajul masinii cu ajutorul caruia eliminam ambiguitatile. Limbajul de programare este limbajul in care scriem instructiunile (*codare*). Este un limbaj **HIGH-LEVEL**, creat si inteles de om, dar care nu este inteles si de computer.

Computerul intelege limbajul masina, **LOW-LEVEL**. Exista doua modalitati principale de transformare a limbajului de programare in limbaj masina, doua tipuri de limbaje si anume:

- **Limbajul compilabil** (exemplul C, C++) – în care o aplicație numită compilator, transformă codul din limbajul de programare (program sursă) în cod mașină. În funcție de mărimea programului compilarea poate fi o operațiune de durată. La orice modificare compilarea trebuie reluată. Compilarea trebuie făcută pentru fiecare tip de mașină (tip de computer/sistem de operare) în parte. Principalul avantaj al programelor compilate este că au o viteză foarte mare de execuție;
- **Limbajul interpretabil** (exemplul **Python**) – nu necesită compilare. Există un interpretor (în cazul Python o mașină virtuală) care transformă ad-hoc instrucțiunile în cod mașină. Principalul avantaj este independența codului de platformă și mașină. Odată instalat interpretorul pe mașină, poate executa orice cod Python. Principalul dezavantaj este viteză mai redusă de execuție.

Limbajul de programare are două componente fundamentale:

- **Sintaxa** - forma în care este scris codul și care ține cont de o serie de reguli simple. Dacă instrucțiunile noastre nu respectă sintaxa vom primi o eroare de sintaxă. În Python mesajele de eroare sunt destul de clare, indicând exact linia de cod unde a apărut eroarea;
- **Semantica** – înțelesul acestuia. Practic, aici intervine arta programatorului.

Instrucțiunile pot să fie corecte din punct de vedere al sintaxei, dar să nu facă ceea ce ne dorim de fapt. Modul în care se succed instrucțiunile, astfel încât programul să facă ceea ce ne așteptăm, tin de semantica.

1.1.4 Programarea eficientă – pași de urmat

1. **Analiza problemei.** În această etapă va trebui să înțelegem cât mai exact care este scopul programului. Acest lucru se realizează în strânsă legătură cu clientul. De cele mai multe ori sunt necesare abilități de comunicare pentru a “scoate” de la client ceea ce este important de știut;
2. **Determinăm specificațiile**, stabilim ce trebuie să facă programul, nu cum trebuie să facă. La acest nivel, de asemenea, este necesară o strânsă colaborare cu clientul. Vom determina care sunt *intrările, ieșirile și relațiile* în cadrul programului;
3. **Algoritmul (design).** Încercăm să definim o soluție care să răspundă specificațiilor, pas cu pas, în pseudocod (un limbaj ad hoc, undeva între limbajul uman și limbajul de programare).

Dacă am creat algoritmul, problema e că și rezolvată, urmând să codăm.

Daca nu am putut crea algoritmul nu inseamna neaparat ca problema nu are solutie. Pot exista solutii la care noi nu ne-am gandit sau pot exista solutii care nu sunt realizabile cu resursele actuale, etc.

4. **Implementarea.** In aceasta etapa se programeaza efectiv. Cu cat etapa precedenta descrie mai clar pasii de urmat cu atat programarea va fi mai usoara.
5. **Testare / Debug.** In aceasta etapa cautam tot ce ar putea face ca programul sa se blocheze sau sa nu functioneze. Aceasta etapa este destul de laborioasa si necesita foarte mult timp. Ar putea fi implicat si clientul sau potentiali clienti, care stiu cel mai bine ce vor sa obtina de la program si astfel pot crea situatii care nu pot fi prevazute de programatori.
6. **Mentenananta.** De regula programele sunt dezvoltate continuu, fiind mapate pe nevoile in continua clientilor.

Desigur, acesti pasi formeaza un ciclu care va fi repetat total sau partial pana la implementarea unei solutii corespunzatoare din toate punctele de vedere.

1.1.5 Python - avantaje si aplicabilitate

Principalele *avantaje* ale Python sunt urmatoarele:

- a. Este un limbaj de programare foarte apropiat de limbajul uman, ceea ce-l face usor de asimilat si-i confera rapiditate.
- b. Are o comunitate foarte bine dezvoltata, care pune la dispozitie o varietate de module (programe care ruleaza independent) si care pot fi integrate total sau partial in propriile aplicatii pe care le dezvoltati. Unele dintre acestea sunt contra cost, insa majoritatea sunt disponibile gratis. Printre site-urile de interes in acest sens putem enumera:
<https://www.python.org/> - site-ul oficial al fundatiei Python si principala sursa de informatii
<https://python-forum.io/> - un forum activ al developerilor Python
<https://plus.google.com/s/python/top> - comunitati Google, unde gasiti si comunitati Python
- c. Este un limbaj procedural, care permite, optional, programarea orientata pe obiecte;

- d. Dispune de un debugger foarte puternic, care identifica toate resursele apelate pana la eroarea aparuta, ceea ce poate fi foarte util in depanarea acesteia.
- e. Este cotate intre primele 5 limbaje utilizate, impreuna cu C++, C, C# si Java;

Aplicabilitatea Python acopera multiple domenii de interes printre le care amintim pe urmatoarele:

- a. Filme de animatie si jocuri (inclusiv Disney, Pixar, etc.);
- b. Cautarea in baze de date (Youtube, Google, Yahoo, etc.);
- c. Testare (HP, Intel, etc.);
- d. Critografie (NSA, etc);
- e. Aplicatii militare (NASA, etc);
- f. Aplicatii de tip ERP/CRM;
- g. Aplicatii WEB;

Desigur, aplicabilitatile sunt multiple si se bazeaza pe modularizare, dupa cum o sa discutam in capitolele urmatoare. Important de stiut este ca aceste module pot fi integrate foarte usor in programe scrise in alte limbaje, cum ar fi C, C++, Java.

1.1.6 Instalarea Python

Pachetul de instalare il puteti descarca de la adresa: <https://www.python.org/downloads/>. Indiferent daca instalati versiunea 2.x sau 3.x pasii sunt asemanatori. Vom prefera versiunea 3.x, pentru ca este singura care se dezvoltă in acest moment. Versiunea 2.x doar se “carpeste”. Unele dintre modulele din versiunea 2.x sunt deja “traduse” si in versiunea 3.x. Altele sunt compatibile pur si simplu (nu au diferente fata de versiunea 2.x). De asemenea, exista un script “2 to 3” care poate traduce codul modulelor din versiunea veche in cea noua. Cursul este mapat pe versiunea 3.5.2.

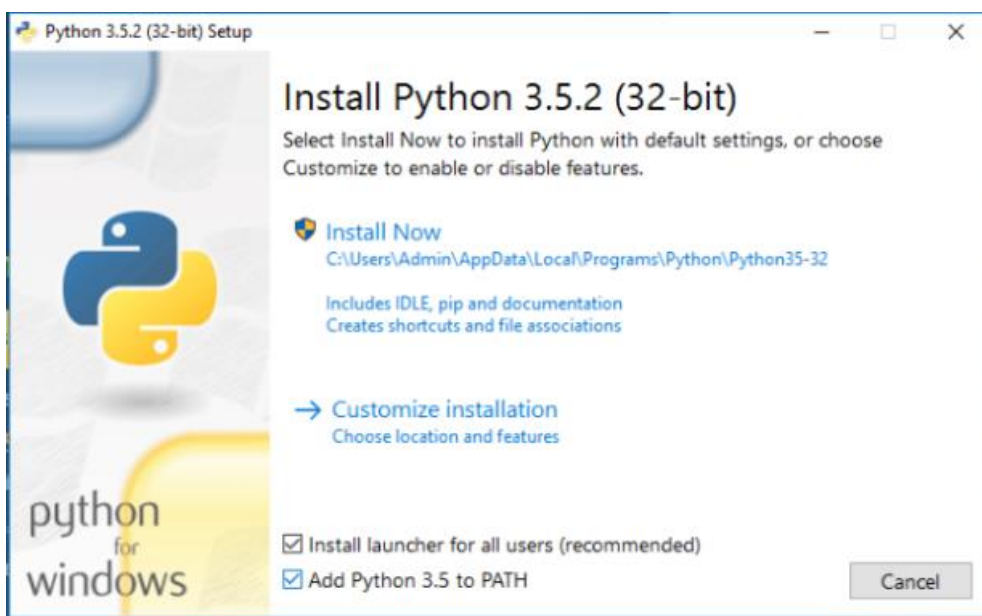
Instalarea in Windows

1. Facem download (vezi imaginea). Va fi descarcat fisierul python-3.5.2.msi, un executabil care va instala python si toate librariile incluse pentru dv.



Exista si instalari alternative. Pentru moment este suficient sa stiti versiunea originala. Pe masura ce avansati veti putea incerca si alte versiuni in functie de necesitati.

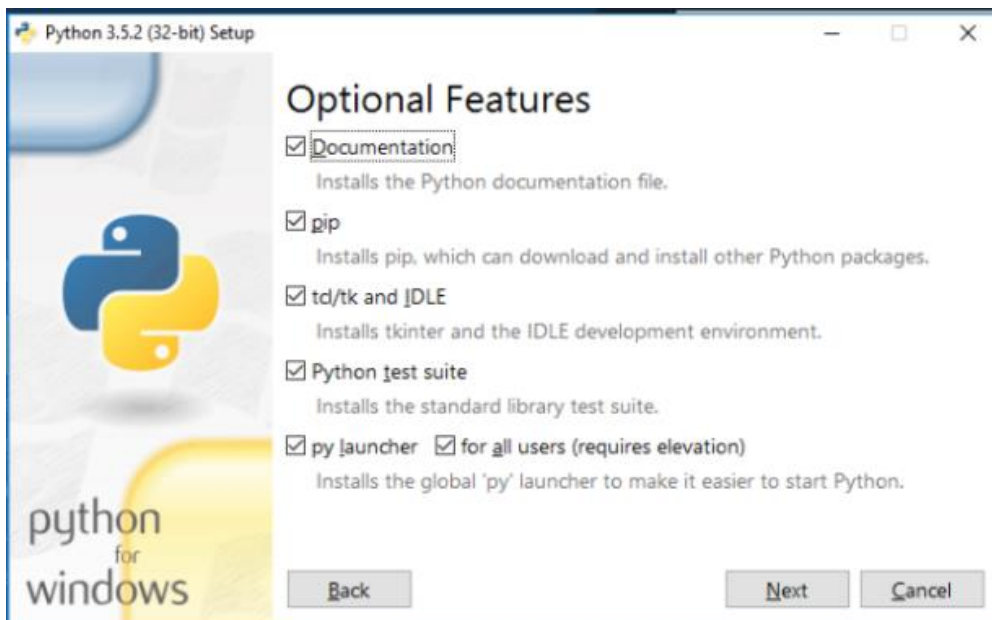
2. Dati click pe executabil, apoi click pe run:



Este important sa dati click pe “Customize installation”. Va va permite sa faceti setari importante.

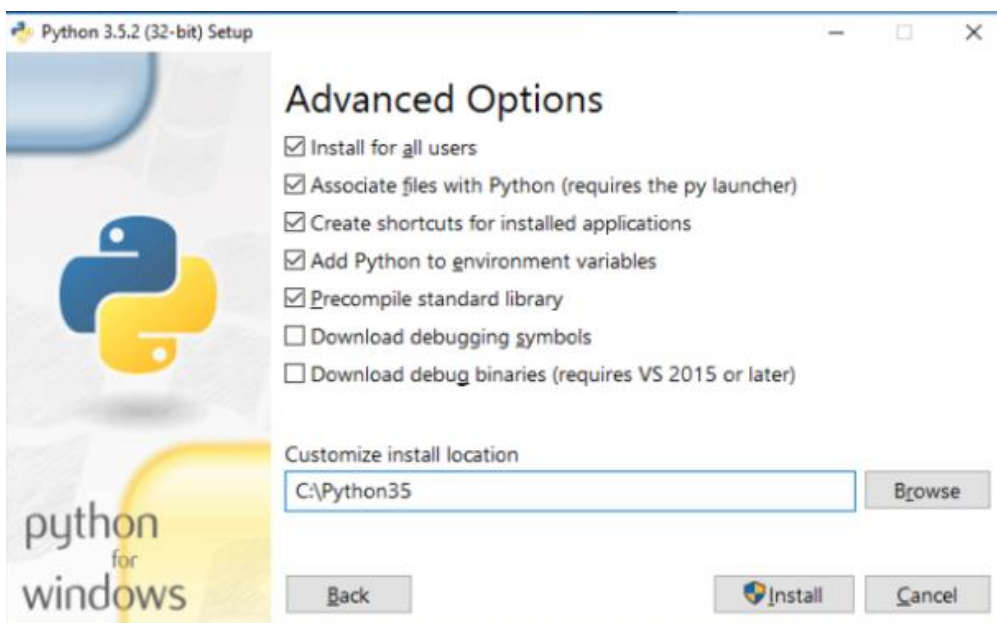
De asemenea, este important sa bifati cele doua optiuni din imagine. Una va permite tuturor utilizatorilor computerului sa ruleze Python. Cea de-a doua va permite accesul mai usor din linie de comanda (acces direct cu executabilul python indiferent de cale, directorul unde ne aflam).

3. Selectati optiunile bifate:



Prima asigura accesul la documentatie. A doua instaleaza pip, cu ajutorul caruia instalam noi module python. A treia si urmatoarele dau acces la diferite resurse python.

4. Selectati calea unde se va instala (unde vor fi scrise librariile python). Este preferabil sa fie cat mai aproape de radacina. C:/Python 3x poate fi o varianta.



5. La un moment dat se va deschide o fereastră de tip command line, in care se vor instala doua instrumente foarte importante: setuptools si pip. Nu trebuie sa faceti nimic, se va inchide singura. Cu ajutorul acestora instalam noi module python.

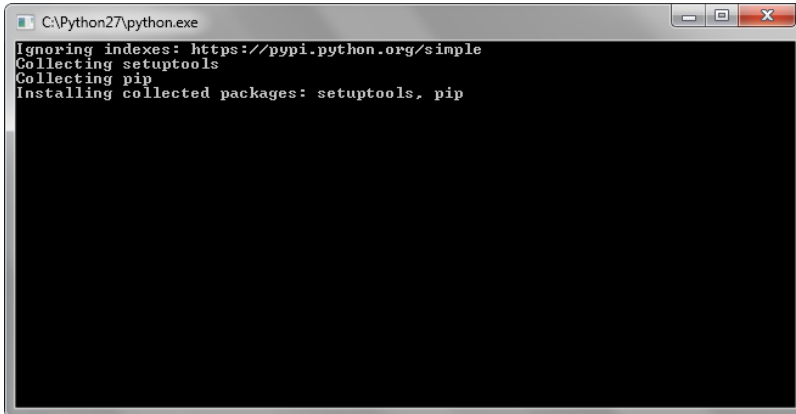
Pentru adaugarea de noi module, se intra in linie de comanda, in directorul unde se afla executabilul modulului (predupune descarcare anterioara), si se scrie instructiunea:

C:/.../python setup.py install

Sau, fara a fi necesar sa descarcam in prealabil modulul

C:/pip install nume_modul

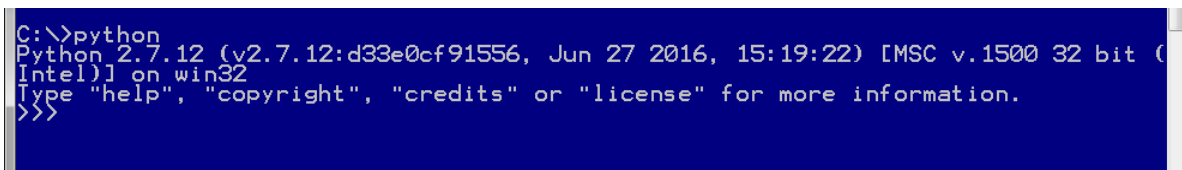
Vom reveni asupra acestor instructiuni la capitolul dedicat modulelor.



```
C:\Python27\python.exe
Ignoring indexes: https://pypi.python.org/simple
Collecting setuptools
Collecting pip
Installing collected packages: setuptools, pip
```

6. Instalarea este completa:

7. Putem intra in linie de comanda si tastam python. Va apareea prompterul >>>, care indica ca interpretorul python este gata de lucru in modul interactiv (asteapta comenzi, una cate una, pe care le va executa):



```
C:\>python
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Instalarea pe alte sisteme de operare

- Pentru ultimele versiuni de MacOS exista Python preinstalat;
- Ultimele versiuni de Linux, distributiile CentOS, Fedora, Redhat Enterprise si Ubuntu vin cu Python preinstalat;
- Pentru instalarea pe alte sisteme de operare veti proceda conform specificului fiecarei platforme. Daca dati click pe download, pe site-ul oficial Python, aveti link-uri catre celelalte platforme.

Documentatia Python este disponibila in Documentation pe acelasi site. Puteti accesa documentatia online sau o puteti descarca pe propriul computer.

In **Community** puteti intra in legatura cu diverse grupuri care au acelasi interes: Python.

In *Success Stories* puteti gasi informatii despre proiecte Python de succes.

1.1.7 Fisiere si executabile Python

Executabilele Python

In directorul unde ati instalat Python, implicit c:/Python35, veti gasi doua executabile:

- ***python.exe*** – care face ca programul sa ruleze in consola. Este preferabil atunci cand nu rulam interfete grafice;
- ***pythonw.exe*** – ruleaza in modul non-consola, preferabil pentru interfete grafice.

Tipuri de fisiere Python

Fisierele Python pot avea urmatoarele extensii:

- ***.py*** - contine cod sursa. Practic, orice script creat de programator poate fi salvat cu aceasta extensie. Fisierele cu aceasta extensie pot fi rulate din consola;
- ***.pyw*** - contine cod sursa. Practic, orice script creat de programator poate fi salvat cu aceasta extensie. Fisierele cu aceasta extensie pot fi rulate in modul non-consola (aplicatii grafice, daemon, etc.);

Aceste doua tipuri de fisiere sunt executabile, avand ca efect rularea scriptului pe care il contin. Python are un ***interpretor*** prin intermediul caruia scriptul este rulat independent de hardware-ul computerului (Python Virtual Machine).

- ***.pyc*** - fisierele cu aceasta terminatie sunt fisiere .py sau .pyw “byte-compiled”, gata de a fi prelucrate de interpretor. Acest tip de fisiere este creat automat la prima rulare a programului, fiind, practic, un intermediar intre codul sursa si interpretor. De fiecare data cand vom rula programul se va verifica automat daca fisierul original cu terminatia .py sau .pyw a fost modificat, caz in care fisierul .pyc va fi rescris.

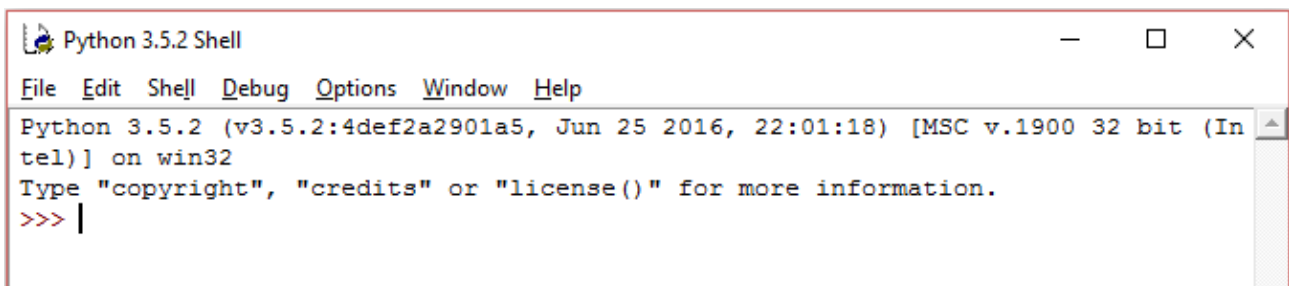
Fisierele .pyc ***ajuta doar la incarcarea mai rapida a modulelor python in memoria RAM a computerului (disponibilitate)***. Timpul de rulare a programului in sine nu este influentat.

- ***.pyd*** - sunt fisiere de tip .dll (Dynamic Link Library), redenumite cu extensia .pyd. Aceste fisiere pot contine executabile, scripturi, drivere, etc. salvarea sub forma .dll si respectiv .pyd se face cu scopul eficientizarii (reutilizarea codului, reducerea spatiului pe HDD, eficientizarea memoriei).

1.1.8 Codul Python. Utilitare

Pentru a programa in Python putem folosi diferite solutii, numite IDE (Integrated Development Environment), seturi de instrumente care ajuta programatorul sa dezvolte aplicatii Python.

Python are preinstalat un astfel de IDE, numit IDLE (Integrated Development and Learning Environment). Daca mergem in grupul de aplicatii Python vom putea apela IDLE. Se va deschide o fereastră (*Python shell*) in care vom putea lucra atat in mod interactiv (interactionam direct cu Python, orice comanda va fi executata imediat) cat si in mod non-interactiv (modul script).



Puteti tasta:

```
>>> print ( "Hello World!" )
```

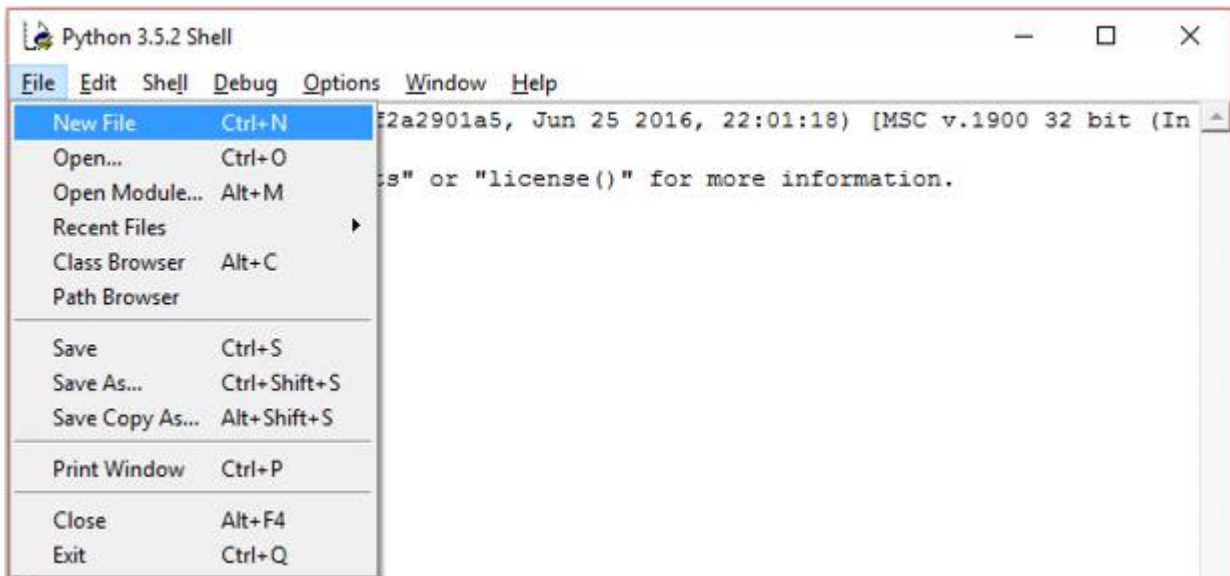
```
Hello World!
```

```
>>> print ( 2 + 2 )
```

```
4
```

Prima expresie va returna Hello World! , salutul traditional al programatorilor, iar a doua rezultatul operatiunii aritmetice de adunare.

Bineinteles, putem edita codul Python in orice editor, inclusiv in Notepad sau in Word. Diferenta intre acestea si IDE este ca cel din urma dispune si de alte instrumente care vin in ajutorul programatorului, cum ar fi: un cod de culori al codului, instrumente de debug, posibilitatea de a deschide ferestre in modul script (File – New window – ca in figura) ceea ce permite atat scrierea cat si deschiderea si rularea scripturilor, alte instrumente utile cum ar fi: incrementarea automata a textului, informatii despre parametrii cu care sunt apelate functiile, metode aplicabile, etc.



Exista si alte IDE pe care le putem utiliza, printre care as aminti doua: **Eclipse** si **PyCharm**.

Primul poate fi descarcat si instalat accesand pagina <https://eclipse.org/>, cel de-al doilea la adresa <https://www.jetbrains.com/pycharm/>. Avantajul acestora este ca ne permit gruparea scripturilor pe proiecte, ofera alte facilitati, cum ar fi lucrul in paralel in consola si intr-o extensie a liniei de comanda, posibilitati suplimentare de debug si formatare a scripturilor, inclusiv indentarea automata, etc.

Desigur, exista si alte solutii si este normal ca fiecare sa aiba propriile preferinte pentru lucrul cu una sau mai multe.

Asa cum am amintit mai devreme, un program python poate fi rulat din consola. Dar poate fi rulat la fel de bine cu un dublu click direct pe fisierul .py sau .pyw. Acesta va fi rulat si va iesi din program cand termina. Pentru a vizualiza output-ul putem introduce, la sfarsitul scriptului urmatoarea instructiune:

```
>>>input("Apasa ENTER pentru a iesi")
```

Ca sa iesim din program nu avem decat sa tastam ENTER.

Exista si un standard, optional, pentru editarea codului Python, **PEP8** (Python Enhancement Proposal). N-o sa utilizam acest standard in curs. Ar fi si dificil sa invatati in paralel Python si acest standard. Este mult mai facil sa-l invatati dupa ce o sa staapaniti notiunile de Python.

Daca doriti mai multe amanunte despre acesta, sau chiar sa-l utilizati, puteti accesa <https://www.python.org/dev/peps/pep-0008/>.

1.2 Primele programe in Python

1.2.1 Lucrul cu siruri de caractere. Primii pasi in Python, functia *print()*

Comentarii

In programare, documentarea este foarte utila. Datorita complexitatii pe care o pot atinge programele documentatia este cheia cautarii rapide in modulele programului pentru update, debug, etc. In Python documentarea se poate face cu ajutorul comentariilor.

Comentariile in Python incep cu caracterul “#”, de la inceputul liniei sau oriunde pe o linie. Incepand de la acest caracter, tot ce scriem pana la sfarsitul liniei va fi ignorat de interpretorul Python.

Primele linii ale unui program Python vor fi, de regula, sub forma de comentarii si contin informatii despre program astfel:

Numele programului

Informatii despre scopul programului

Creator – Data – Versiune – alte informatii utile

Comentarii pe mai multe linii pot fi scrise cu ajutorul ghilimelelor triple:

Functia *print()*

Cu ajutorul acestei functii vizualizam rezultatul instructiunilor Python. Este case sensitive, va fi scrisa cu litere mici si va cuprinde intre paranteze expresia de printat. Aceasta tine de sintaxa. Dupa cum o sa vedeti si in continuare, nerespectarea sintaxei va genera o eroare.

Dupa cuvantul cheie print va urma, intre paranteze rotunde, un sir de caractere sau o expresie. Sirurile de caractere vor fi cuprinse intre **ghilimele** sau **apostroafe**. Cu ce incepem trebuie sa si terminam, in caz contrar va fi generata o eroare.

In versiunea 2.x sirul de caractere sau expresia pot fi cuprinse sau nu intre paranteze rotunde.

```
>>>print ('Hello World!')  
Hello World!
```

Daca textul in sine trebuie sa cuprinda ghilimele vom putea utiliza apostroafe si invers:

```
>>>print ('Putem folosi "Ghilimelele" ')
Putem folosi "Ghilimelele"
```

Ghilimelele triple utilizate cu print() permit scrierea unui text pe mai multe linii si vizualizarea acestuia exact cum a fost scris. Putem folosi ghilimele sau apostroafe. In Python exista notiunea de *linie fizica*, ce vedem cand scriem instructiunea pe o linie si *linie logica*, o instructiune (statement) care se poate intinde pe mai multe linii fizice.

```
>>>print ("""Este preferat sa nu depasiti marginea din dreapta a ecranului
in niciun caz nu mai mult de 79 de caractere, asa ca, pentru vizualizarea unui text,
puteti continua pe linia urmatoare folosind ghilimelele triple
""")
```

Este preferat sa nu depasiti marginea din dreapta a ecranului
in niciun caz nu mai mult de 79 de caractere, asa ca, pentru vizualizarea unui text,
puteti continua pe linia urmatoare folosind ghilimelele triple

Back-slash “\ ” permite continuarea unei instructiuni print, care are ca parametru un sir de caractere, pe mai multe linii. Desi instructiunea este partajata pe mai multe linii ea va fi tratata ca un tot unitar.

```
>>>print ('Invat Python!' \
' Faceti liniste...')
Invat Python! Faceti liniste...
```

Punct si virgula (semicolon) (“ ; “) este caracterul cu ajutorul caruia scriem mai multe instructiuni pe aceeasi linie. Rezultatul fiecarei instructiuni va fi printat pe o linie separata.

```
>>>print ('Astazi e ziua ta'); print ("Zi frumoasa ca tine"); print ('Ti-am adus
trandafiri')
Astazi e ziua ta
Zi frumoasa ca tine
Ti-am adus trandafiri
```

Virgula ne permite sa scriem doua sau mai multe expresii ca parametri ai functiei print. Aceasta modalitate este utila pentru combinarea diferitelor tipuri de date astfel:

```
>>>print ('Invat Python!', 'Faceti liniste...', 2 + 3, 'minute')
Invat Python! Faceti liniste... 5 minute
```

1.2.2 Concatenarea si repetitia sirurilor de caractere

Concatenarea sirurilor de caractere se poate face cu operatorul “ + “. Atentie, concatenarea poate fi facuta doar intre siruri de caractere. Daca vrem sa concatenam un sir de caractere cu un numar va fi generata o eroare. Totusi, putem transforma numarul intr-un sir de caractere, cu ajutorul functiei str() si concatenarea va fi astfel posibila. Concatenarea se efectueaza fara eventuale spatii intre siruri. Daca avem nevoie de spatii sau alte caractere intre acestea avem posibilitatea sa le includem la capetele sirurilor sau sa le concatenam separat.

```
>>>print ('Alina' + ' ' + 'viseaza!')    # spatiu concatenat separat
Alina viseaza!
```

sau

```
>>>print ('Alina ' + 'viseaza!')          # spatiu inclus dupa Alina
Alina viseaza!
```

Repetitia sirurilor de caractere se poate face cu operatorul “ * “.

```
>>>print (3 * 'Gooooool!')
Gooooool! Gooooool!Gooooool!
```

Putem combina concatenarea si repetitia:

```
>>>print ('Tra' + 4 * ', la')
Tra, la, la, la, la
```

```
>>>print ('Sunati la 07' + 4 * '90')
Sunati la 0790909090
```

Atentie, doar operatorii + si * pot fi utilizati pentru concatenare si repetitie. Ceilalti operatori aritmetici nu au niciun efect asupra sirurilor de caractere.

1.2.3 Secvente de evadare intr-un sir de caractere, utilizat in functia print()

Secventele de evadare sunt caractere speciale, formate printr-o combinatie de caractere normale utilizate in functia print, care au o semnificatie proprie, diferita de caracterele ce o compun, astfel:

- **\a** – bip sau caracter grafic. Dacă rulăm fișierul prin dublu click sau din linie de comandă vom obține un bip. Dacă-l rulăm dintr-o aplicație grafică sau IDE va genera un semn grafic (bulina, pătratul, etc);
- **\t** – tab. Are ca efect plasarea unui “tab”, echivalentul a patru spații, în locul respectiv. Mai multe caractere de acest gen înseamnă mai multe tab-uri sirul în cauză;
- **\caracter** – va tipări caracter. Este util când vrem să tipărim caractere care altfel au altă semnificație (slash, back-slash, ghilimele, \$, etc);
- **\n** – sare la rândul următor. Practic, este terminator de linie. Dorim să adăugăm mai multe rânduri libere, îl folosim de mai multe ori.

Fiecare dintre aceste secvențe de evadare sunt tratate ca un singur caracter, deși au în componență mai multe caractere, lucru foarte important când vom utiliza indici sau dacă dorim să știm câte caractere conține sirul nostru.

1.2.4 Lucrul cu numere

În Python avem trei tipuri de numere: *întregi (int)*, *rationale (float)* și *complexe*. În funcție de tipul numerelor rezultatul operațiilor aritmetice diferă foarte mult. De asemenea, utilizarea operatorilor și implicit rezultatul împărțirii diferă între versiunile 2.x și 3.x. În versiunea 2, având în vedere că numerele întregi au o plajă de valori foarte restrânsă, între -2^{31} și $2^{31} - 1$, există și tipul de numere întregi *long*, care au litera “L” la sfârșitul numărului (preferabil litera mare, pentru a nu se confunda cu cifra unu). Acest tip dispăre în versiunea 3.x fiind unificat cu numerele de tip *int*, care vor suporta și valori mai mari.

Adunarea(+)

```
>>>print (5 + 5)      # Adunarea a doua numere intregi
10
>>>print (5 + 5.0)    # Adunarea a doua numere dintre care cel puțin unul este float
10.0
```

Scaderea(-)

```
>>>print (5 - 1)      # Scaderea a doua numere intregi
4
>>>print (5.0 - 3.0)  # Scaderea a doua numere dintre care cel puțin unul este float
2.0
```

Inmultirea()*

```
>>>print (5 * 5)      # Inmultirea a doua numere intregi
25
>>>print (5 * 2.0)    # Inmultirea a doua numere, cel putin unul este float
10.0
```

Impartirea(/)

```
>>> print (23 / 4)      # Impartirea
5.75
>>> print (23 // 4.0)   # Impartirea a doua numere intregi (doar catul impartirii)
5.0
```

In versiunea 2.x sunt functionalitati diferite pentru impartire.

```
>>>print 20 / 5        # Impartirea exacta a doua numere intregi
4
>>>print 23 / 4        # Impartirea a doua numere intregi (doar catul
5
>>>print 23 / 4.0      # Impartirea a doua numere, cel putin unul este float
5.75
```

Modulo (%)

```
>>>print (23 % 4)      # Restul impartirii
3
```

*Ridicarea la putere (**)*

```
>>>print (9 ** 3)
729
```

Numere complexe (a + bj)

```
>>>print ( (2 + 2j) * (2 + 4j) )
(2 + 6j)
>>>print ( (2 + 2j) / (2 + 4j) )
(0.6 - 0.2j)
>>>print ( (2 + 2j) + (2 + 4j) )
(4 + 6j)
```

```
>>>print ( (2 + 2j) - (2 + 4j) )  
-2j
```

In cazul numerelor complexe este folosit “j” pentru partea imaginara a numerelor, nu “i” cum se studiaza la matematica. Atentie, indicatorul lui j trebuie scris chiar daca este 1.

Precedenta operatorilor in Python:

- Parantezele rotunde (numai parantezele rotunde pot fi utilizate in expresii matematice, parantezele drepte si acoladele avand alte utilizari);
- Ridicarea la putere;
- Inmultirea, impartirea, modulo;
- Adunarea si scaderea;
- De la stanga la dreapta.

Operatii cu mai multi operatori

```
>>>print ( ( (7 + 8) * 4) / (2 + 4) )  
10
```

Conversii de numere

```
int('100', 16)      # transforma un string in numar decimal, specificand baza  
256                 # in care este exprimat (ex. Din baza 16 in baza 10)  
                    # daca este exprimat in baza 10 nu este necesara mentionarea
```

```
float('10')         # transforma un string in float  
10.0
```

```
int('2.5')          # eroare in cazul in care stringul contine si o parte rationala
```

Traceback (most recent call last):

File "<pyshell#17>", line 1, in <module>

```
int('2.5')
```

ValueError: invalid literal for int() with base 10: '2.5'

```
int(float('2.5'))   # solutia este transformarea in float si float in intreg. Partea  
2                   # zecimala va fi ignorata
```

1.2.5 Lucrul cu variabile

Valorile fixe ale numerelor sau stringurilor sunt constante. Valoarea lor este si ramane neschimbata.

Exemple de constante numerice:

```
>>>print (100)
100
>>>print (15.87)
15.87
```

Exemplu de constanta string:

```
>>>print ('Hello World!')
Hello World!
```

Variabila este un spatiu de memorie, care primeste un nume si stocheaza o valoare.

Spre deosebire de alte limbaje de programare, in care se seteaza un tip de date pentru fiecare variabila, chiar la momentul declararii acesteia, in Python nu se intampla acest lucru, aceeasi variabila putand stoca succesiv diferite tipuri de date.

Numele variabilelor pot contine litere, cifre si underscore (_). Numele este *case sensitive* si poate incepe doar cu o litera sau underscore. Implicit, nu poate fi format doar din cifre.

Stiluri de denumire:

```
>>>NotaMax = 100,    in care folosim combinatii de cuvinte lipite, incepand cu litere mari;

>>>nota_max = 100,   in care folosim litere mici si combinatii de cuvinte despartite de underscore.
```

Cuvinte rezervate, care nu pot fi folosite pentru denumirea variabilelor:

and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, yield, while, with

Cu ajutorul variabilelor informatia poate fi stocata si manipulata.

Exista *variabile globale*, care sunt vizibile (utilizabile) in tot programul si care se declara la inceputul programului, imediat dupa comentariile care explica scopul programului.

Exista *variabile locale*, care sunt vizibile intr-un spatiu mai restrans (de exemplu in interiorul unei functii).

Atribuirea de valoare variabilelor, exemple

```
>>>x = 5                # atribuire de valoare unei variabile
```

```
>>>luni = 'Monday'
```

sau

```
>>>x, y, z = 1, 20, 300  # atribuire de valoare mai multor variabile, cu aceeasi  
instructiune
```

```
>>>x = x + 5            # atribuire de valoare printr-o expresie
```

Atribuire prin metoda *augmentation*.

```
>>>x += 5               # incrementare, echivalent cu x = x + 5
```

```
>>>x -= 5               # decrementare, echivalent cu x = x - 5
```

```
>>>x *= 7               # inmultire, echivalent cu x = x * 7
```

```
>>>x /= 7               # impartire, echivalent cu x = x / 7
```

```
>>>x = None             # initializeaza o variabila, fara nicio valoare
```

Unele dintre acestea opereaza si asupra sirurilor de caractere (“+” si “*”).

1.2.6 Captarea unui sir de la tastatura

input() este o functie cu ajutorul careia utilizatorul poate introduce siruri de caractere (stringuri) de la tastatura. In versiunea 2.x se utilizeaza *raw_input* cu acelasi scop. In aceeasi versiune input are o alta semnificatie (importa doar numere);

Putem utiliza o variabila pentru a capta un sir de caractere:

```
>>>nume = input("Introduceti numele: ")
```

Putem utiliza input pentru a iesi controlat dintr-un script Python:

```
>>>input("Apasa 'ENTER' pentru a iesi!")
```

Aceasta functie va capta intotdeauna siruri de caractere. Pentru a capta numere vom face o transformare din string in numar:

```
>>>numar_int = int( input("Introduceti numele: ") )
```

sau

```
>>>numar_float = float( input("Introduceti numele: ") )
```

De asemenea, putem utiliza *eval(input())* pentru a capta direct numere de la tastatura.

1.3 Recapitulare

1.3.1 Recapitulare

Python este un limbaj interpretabil si byte-compiled;

In Python, programarea orientata pe obiecte este optionala;

Python permite detectarea foarte precisa a erorilor aparute;

Toti identificatorii (variabile, functii, clase, etc) si toate datele de tip string sunt case sensitive;

Sirurile de caractere vor fi mentionate intotdeauna intre ghilimele/apostroafe;

Nu confundam un sir de caractere nu un numar, chiar daca in componenta lui sunt doar cifre. Consecinta, inainte de a evalua o expresie, evaluati mai intai tipurile de date;

Secventele de evadare ne ajuta sa obtinem diferite efecte prin combinatii de caractere;

Putem folosi ghilimele triple pentru comentarii pe mai multe linii;

Putem folosi print() cu ghilimele triple, pentru a vizualiza un text exact asa cum a fost scris;

Putem face transformari intre tipurile de date, in anumite conditii;

Facem distinctia intre operatorul plus aplicat numerelor sau stringurilor;