Materiale utile seminar

Python - elemente de baza
Print/Whitespace/Control Flow/Import

- a = 2
- b = 3
- **print**(a + b)

- a = 2
- b = 3
- x = - b/a
- print(x)

- a = 2
- **if a % 2 == 0:**
  - print(“par”)
- **else:**
  - print(”impar”)

- **from math import sqrt**
- a = 1
- b = 4
- c = 4
- delta = b*b - 4*a*c
- x1 = (-b + sqrt(delta)) / (2*a)
- x2 = (-b - sqrt(delta)) / (2*a)

- a = 5
- **while a > 0:**
  - print(a)
  - a = a - 1

Python - elemente de baza
List(create;append;pop;idx)/String(len)

# Liste 	
xs = **[]**
xs.**append**(1)
xs.**append**(2)
xs.**append**(3)
 	
print(xs)
	# Liste
	x = xs.**pop**()
	print**(x, xs)**
	
	xs = xs **+** [4, 5, 6]
	
	print(xs, xs**[0]**, xs**[1]**)
		# Liste
		x = xs.pop(**0**)
		y = xs.pop(**0**)
		print(x, y, xs)
		
		print(**len**(xs))
			# Liste
			for x in xs:
			  print(x)
			
			xs.**extend**([4, 5, 6])
			for x in **reversed**(xs):
			  print(x)

# String-uri 	
s = **“**Hello, world!**”**
print(s)
 	
s_length = **len**(s)
for x in **range**(s_length):
  print(s[x])
	# String-uri
	for c in s:
	  print(c**, ‘___’**)
	
	for c in **reversed**(s):
	  print(c + “a”)
	print(s)
		# String-uri
		multi = **“””** String
		special pe mai multe
		randuri **“””**
		
		(REPL: multi,
		print(multi))
			x = 19391
			s = str(x)
			r = **list**(reversed(s))
			
			for i in range(len(s)):
			  if s[i] != r[i]:
			  print(“nu”)
			print(“da”)

Python - elemente de baza
Functii(def; param; return; apel)/Tuplu

```
# Functii
def suma(a, b):
        c = a + b
        return c

print(suma(1, 3))
```

```
# Functii
def suma_lista(xs):
        s = 0
        for x in xs:
                s += x
        return s
```

```
# Functii
def cauta_nr(n, xs):
        for x in xs:
                if n == x:
                        return True
                else:  return False
```

```
# Functii
def aduna_val(x, val=3):
        return x + val

print(aduna_val(1, 1))
print(aduna_val(1))
```

```
# Tuplu
a = (1, 2, 3)

# Liste de tupluri
n = 60
xs = [(2, 2), (3, 1), (5, 1)]
```

```
def fp(n):
        xs = []
        .....
        return (len(xs), xs)
(n, lista) = fp(60)
print("Nr. perechi:", n, "lista:", lista)
```

Recapitulare Seminar I

```
if n%5 == 0:
        m = n + 5    # (1)
        print(m)     # (2)
else:
        print(False)
```

```
while n:
        print(n, '\t', n - 1)
        n -= 1
print(n, end="\n")
```

```
for i in range(1, n+1):
        print("i =", i)
# range(n) -> 0, 1, 2 ... n - 1
# range(i,s) -> i, i+1, i+2 ... s-1
# range(i,s,p)-> i, i+p, i+2*p,...s-1
```

```
xs = []
xs.append(1)# xs = [1]
xs.append(2)# xs = [1, 2]

n = len(xs)
```

```
x = xs.pop()# x=2, xs=[1]
y = xs.pop(0)#y=1, xs=[]

for x in xs:
        print(x)
```

```
def functie(param1, param2):
        cs = [param1, param2]
        s = 0
        for c in cs:
                s += c
        return s
```

Numere binare
Reprezentare (1)

- Reprezentarea interna **n = 14** => format binar
- Conversie baza 10 -> baza 2:
  - se imparte numarul **n** la 2 **cu (cat, rest)**
  - se memoreaza **restul**
  - **n** devine **catul**
  - se repeta procedeul pana cand **n devine 0**
  - numarul in baza 2 este dat de **resturi (in ordine inversa)**

Numere binare
Reprezentare (2)

- Reprezentarea interna **n = 14** => format binar
- Exemplu:
  - 14 : 2 = 7 rest 0 ^
  - 7 : 2 = 3 rest 1 |
  - 3 : 2 = 1 rest 1 |
  - 1 : 2 = **0** rest 1 |
- $(14)_{10} = (1110)_2$

Numere binare
Reprezentare (3)

- b = 1110
- Conversie baza 2 -> baza 10:
  - se inmulteste fiecare cifra cu $2^{pozitie}$ si se aduna rezultatele
  - poz = [0, 1, 2, 3]
  - **r**_b = [0, 1, 1, 1] # numarul este inversat
  - $(1110)_2 = 0 * 2^0 + 1 * 2^1 + 1 * 2^2 + 1 * 2^3$

Numere binare Reprezentare (4)

- Alternativ (b = 1110):
- ←←←←←←←←←←
- 2^3 2^2 2^1 2^0 *
- 1 1 1 0
- 8 + 4 + 2 + 0 = 14

Operatii elementare biti
OR, AND, XOR, NOT

OR:

11000 |
00011 =
11011

AND:

11011 &
01001 =
01001

XOR:

11010 ^
11101 =
00111

NOT:

~ 11001 =
00110

Operatii elementare biti
Shiftare, Verificare, Setare

- 1 << n = “plecand de la 0, punem 1 pe pozitia n a nr in binar)”
  - 1 << **4** = 00000000 → 000**1**0000 (indexare de la 0)
- 16 >> n = “plecand de la 16, mutam numarul n pozitii in dr.”
  - 16 >> 2 = **000**1**0**000 → 00**000**1**0**0 (zero nu conteaza)
- n & (1 << k) = “este bit-ul k setat in n? daca da, rezultatul != 0)
  - 1**1**10 & 0**1**00 = 0**1**00 (operatie bit cu bit), rezultat 4 != 0
- n | (1 << k) = “setam bit-ul k la 1 in n”
  - 1110 | 0001 = 1111 (operatie bit cu bit, rezultat 15)

Verificare manuala operatii biti
Exemplu: verificare bit setat in numar

- Daca vreti sa verificati cum functionaza operatiile pe biti:
  - avand cele doua numere (n **si** (1 << k)) se convertesc in baza 2 in liste din Python si se afiseaza pe ecran (n = 14, k = 2) **&**
  - n = [1, 1, 1, 0]
  - k = [0, 1, 0, 0]
  - rez = [1 and 0, 1 and 1, 1 and 0, 0 and 0]
  - rez = [False, True, False, False] **(and)**
  - rez = [0, 1, 0, 0] **(&)**