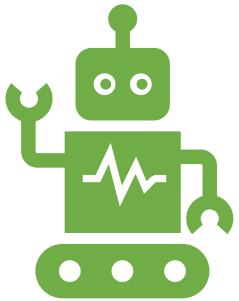
Универзитет „Св. Кирил и Методиј“ во Скопје
ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Вовед во Python



Аудиториски вежби по курсот
Вештачка интелигенција

2019/2020

Python

- Open source јазик со општа намена
- Објектно ориентиран, процедурален, функционален
- Компатибилен со C/ObjC/Java/Fortran
- Компатибилен со C++ (via SWIG)
- Интерпретабилен јазик
- Интерактивна околина

Особини на Python

- **Лесен за учење** – има релативно малку клучни зборови, едноставна структура, јасно дефинирана синтакса
- **Портабилен** – може да се извршува на голем број различни хардверски платформи и го има истиот интерфејс на сите нив
- **Поседува обемна стандардна библиотека** – исто така портабилна
- **Проширлив** – може да се додаваат модули на ниско ниво на Python интерпретерот
- **Поддршка за бази на податоци** – нуди интерфејси кон сите поголеми комерцијални бази на податоци
- **Поддршка за GUI програмирање**

Особини на Python (2)

- **Поддршка за различни стилови на програмирање** – структурно, функционално, ОО програмирање
- **Скриптен јазик** – може да се користи како скриптен јазик, или да се компајлира во бајт-код за изградба на големи апликации
- **Динамички податочни типови** – поседува високо-нивовски динамички податочни типови и поддржува динамичко определување на типовите
- **Собирање на „ѓубре“** – поддржува автоматско собирање на „ѓубрето“ (garbage collection)
- **Интегрибилен** – лесно може да се интегрира со C, C++, Java, …

Python интерпретер

- **Интерпретер** – програма која чита и извршува код, вклучувајќи изворен код, компајлиран код и скрипти.
- Интерпретерите и компајлерите се слични, затоа што знаат да препознаат и процесираат програмски код. Но:
  - компајлерот го преведува програмскиот код во машински код, кој може да се извршува од оперативниот систем како извршна програма;
  - интерпретерот го заобиколува процесот на компајлирање и го извршуваат кодот директно.
- Програмскиот код побарува интерпретер за да се изврши. Затоа, овој код без интерпретер повеќе е обичен txt фајл, отколку извршна програма.

Python интерпретер (2)

- Конзолен интерфејс на Python

```
% python
Python 3.6.5 (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Python интерпертерот евалуира изрази:

```
>>> 3*(7+2)
27
```

Основи

Пример

```
x = 34 - 23   # Comment
y = "Hello"   # Another comment
z = 3.45

if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + "World"  # String concatenation
    print(x)
    print(y)
```

Основни конвенции

- Доделување вредност со = и споредба на вредности со ==.
- Операции за броеви + - * / %.
- За спојување (конкатенација) на стрингови +.
- Употреба на % за формирање на стринг при печатење (како кај printf во C).
- Логичките оператори се зборови (and, or, not), не симболи.
- Основната наредба за печатење е функцијата print().
- Првото доделување на вредност на променливата ја креира таа променлива.
  - Не треба да се декларираат типовите на променливите.
  - Python ги определува според вредностите кои им се доделени при првото доделување на вредност.

Основни типови на податоци

- Integer (се претпоставува за броеви)

```
z = 5 / 2  # Answer is 2.5, division
z = 5 // 2  # Answer is 2, integer division
```

- Float

```
x = 3.456
```

- Стрингови
  - Може да се употребуваат двојни " " или единечни ' ' наводници

```
"abc" 'abc'
```

  - Неспарени наводници може да се среќаваат во стрингот

```
"mat's"
```

  - Со три двојни наводници се дефинираат стрингови кои се во повеќе линии или стрингови кои може да содржат и единечни и двојни наводници во нив

```
"""a'b"c"""
```

Аритметички оператори

Оператор	Опис	Пример (a=10, b=21)
+ Собирање	Ги додава вредностите на било која страна од операторот.	a + b = 31
- Одземање	Ја одзема вредноста на левиот операнд од десниот операнд.	a – b = -11
* Множење	Ги множи вредностите на двете страни од операторот	a * b = 210
/ Делење	Го дели операнд со десниот операнд	b / a = 2.1
% Модул	Го дели левиот операнд со десниот операнд и го враќа остатокот	b % a = 1
** Експонент	Се изведува степенување на операндите	a**b =10 на степен 21
// Целобројно делење	Делење на операндите каде што резултатот е количник во кој се занемаруваат цифрите после децималната точка. Но ако еден од операндите е негативен, резултатот е заокружен од нула (кон негативна бесконечност)	9//2 = 4, 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Оператори за доделување

Оператор	Опис	Пример (a=10, b=20)
=	Доделува вредности од операндите на десната страна на операндот од левата страна	c = a + b доделува вредност од a + b на c
+=	Ги собира десниот и левиот операнд и резултантната вредност ја доделува на левиот операнд	c += a е еквивалентен со = c + a
-=	Го одзема левиот операнд од десниот операнд и резултантната вредност ја доделува на левиот операнд	c -= a е еквивалентен со c = c - a
*=	Ги множи левиот и десниот операнд и резултантната вредност ја доделува на левиот операнд	c *= a е еквивалентен со c = c * a
/=	Го дели левиот операнд со десниот операнд и резултантната вредност ја доделува на левиот операнд	c /= a е еквивалентен со c = c / a
%=	Се бара модулот од двата операнда и резултантната вредност ја доделува на левиот операнд	c %= a е еквивалентен со c = c % a
**=	Се изведува степенување на операндите и резултантната вредност ја доделува на левиот операнд	c **= a е еквивалентен со c = c ** a
//=	Целобројно делење на операндите и резултантната вредност се доделува на левиот операнд	c //= a е еквивалентен со c = c // a

Оператори за споредба

Оператор	Опис	Пример (a=10, b=20)
==	Ако вредностите на двата операнда се исти, тогаш условот станува вистинит (True)	(a == b) не е вистина
!=	Ако вредностите на двата операнда не се исти, тогаш условот станува вистинит	(a!= b) е вистина
>	Ако вредноста на левиот операнд е поголема од вредноста на десниот операнд, тогаш условот станува вистинит	(a > b) не е вистина
<	Ако вредноста на левиот операнд е помала од вредноста на десниот операнд, тогаш условот станува вистинит	(a < b) е вистина
>=	Ако вредноста на левиот операнд е поголема или еднаква на вредноста на десниот операнд, тогаш условот станува вистинит	(a >= b) не е вистина
<=	Ако вредноста на левиот операнд е помала или еднаква на вредноста на десниот операнд, тогаш условот станува вистинит	(a <= b) е вистина

Логички оператори и оператори за припадност

Оператор	Опис	Пример (a=True, b=False)
and Логичко И	Ако двата операнди се вистините тогаш условот станува вистинит	(a and b) is False
or Логичко ИЛИ	Ако било кој од двата операнди е вистинит (non-zero) тогаш условот станува вистинит	(a or b) is True
not Логички комплемент	Се употребува за да се смени логичката состојба на операндот	Not(a and b) is True

Operator	Description	Пример
in	Се евалуира во True ако ја најде променливата во специфицираната секвенца, инаку False	x in y, here in results in a 1 if x is a member of sequence y
not in	Се евалуира во True ако не ја најде променливата во специфицираната секвенца, инаку False	x not in y, here not in results in a 1 if x is not a member of sequence y

Оператори за идентитет

Оператор	Опис	Пример
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, тука **is** резултира во 1 ако id(x) е еднакво на id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, тука **is not** резултира во 1 ако id(x) не е еднакво на id(y).

Бит оператори

Оператор	Опис	Пример (a=0011 1100, b=0000 1101)
& Бинарно И	Операторот го копира битот во резултатот, ако постои и во двата операнди	(a & b) (значи 0000 1100)
| Бинарно ИЛИ	Го копира битот, ако постои во еден операнд	(a | b) = 61 (значи 0011 1101)
^ Бинарно XOR	Го копира битот, ако е сетиран во еден операнд, но не и во двата	(a ^ b) = 49 (значи 0011 0001)
~ Бинарен Комплемент	Унарен оператор со ефект на променување на битовите	(~a) = -61 (значи 1100 0011 во форма на двоен комплемент поради бинарниот број со знак
<< Бинарно Лево Поместување	Вредноста на левиот операнд е поместена на лево за број на битови специфицирани од десниот операнд	a << 2 = 240 (значи 1111 0000)
>> Бинарно Десно Поместување	Вредноста на левиот операнд е поместена на десно за број на битови специфицирани од десниот операнд	a >> 2 = 15 (значи 0000 1111)

Приоритет на оператори

No.	Оператор	Опис
1	**	Степенување
2	~ + -	Комплемент, унарен плус и унарен минус (имињата на методите на последните две се +@ и -@)
3	* / % //	Множење, делење, модул и целобројно делење
4	+ -	Собирање и одземање
5	>> <<	Десено илево бинарно поместување
6	&	Бинарно ‘И'
7	^ |	Бинарно ексклузивно ‘ИЛИ’ и обично ‘ИЛИ'
8	<= < > >=	Оператори за споредба
9	<> == !=	Оператори за еднаквост
10	= %= /= //= -= += *= **=	Оператори за доделување
11	is is not	Оператори за идентитет
12	in not in	Оператори за пропрадност
13	not or and	Логички оператори

Празен простор (Whitespace) (ВАЖНО)

- Празниот простор има значење во Python: особено порамнувањето и новите линии
- Различно во однос на многу други јазици
- Нова линија означува крај на наредба!!
  - Со \ може да се продолжи наредбата во повеќе линии
- Не се потребни загради за да се означат блокови на код
- Наместо тоа, се употребува конзистентно порамнување!
  - Првата линија со помало вовлекување (indentation) е надворешниот блок
  - Првата линија со поголемо вовлекување (indentation) започнува вгнезден блок
- Често две точки (:) се појавуваат на почетокот на нов блок (пр. за декларција на класи и функции)

Коментари

- Коментарите започнуваат со # – остатокот од линијата се игнорира
- Може да се постават “документациски стрингови” како прва линија на која било нова функција или класа што се дефинира
- Ова се употребува од развојните околини, дебагерот и други алатки и е добар стил да се поставува документациска линија:

```
def my_function(x, y):
    """Ova e dokumentaciskiot
    string. Ovaa funkcija sluzhi za
    bla bla bla."""
    # Kodot na funkcijata sleduva ovde...
```

Доделување вредности

- *Доделување на вредност на променлива во Python значи поставување на име (обична променлива) да содржи референца до некој објект.*
  - Доделувањето креира референци, а НЕ копии
- Имињата немаат подразбирлив (default) тип. Објектите имаат типови.
- Python го определува типот на референцата автоматски во зависност од типот на објектот до кој пристапува референцата.
- Се креира променлива првиот пат кога ќе се појави од левата страна на некој израз за доделување на вредност:

```
x = 3
```

- Референците се бришат со garbage collector откако сите променливи кои се поврзани со нив ќе излезат надвор од доменот (out of scope)

Пристапување до непостоечки променливи

- Ако се пристапи име пред да се креира (со поставување на левата страна на доделување) се добива грешка.

```
>>> y
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    y
NameError: name 'y' is not defined
>>> y = 3
>>> y
3
```

Повеќекратно доделување

- Може да се доделуваат вредности на повеќе променливи истовремено.

```
>>> x, y = 2, 3
>>> x
2
>>> y
3
```

Правила за именување на променливи

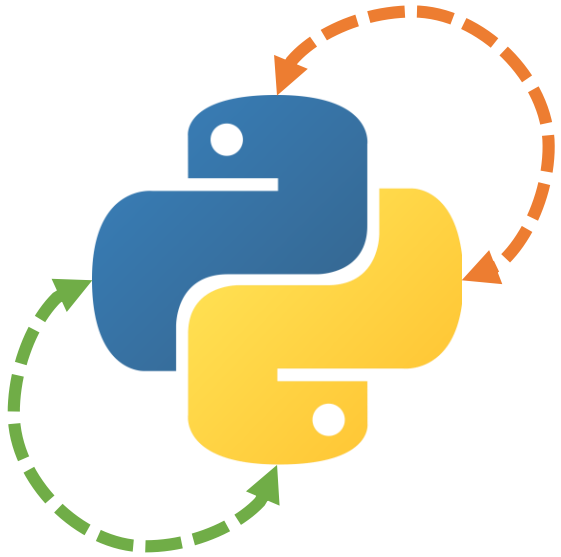
- Имињата на променливите се осетливи на големината на буквите (case sensitive) и не може да започнуваат со цифра
- Може да содржат букви, цифри и долни црти (underscores).

```
bob Bob _bob _2_bob_bob_2 BoB
```

- Постојат резервирани зборови кои не може да се употребуваат:

```
and, 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, while
```

Семантика на референците во Python



Семантиката на референците

- Доделувањето манипулира со референцата
  - x = y **не прави копија** од објектот кој го референцира y
  - x = y прави x **да го референцира** истиот објект кој го референцира и y
- Корисно, но треба да сме внимателни!
- Пример:

```
>>> a = [1, 2, 3]     # a ја референцира листата [1, 2, 3]
>>> b = a             # b сега референцира тоа што a референцира
>>> a.append(4)       # ова ја променува листата која ја референцира a
>>> print(b)          # ако се испечати тоа што b референцира
[1, 2, 3, 4]          # се гледа дека и b е променето
```

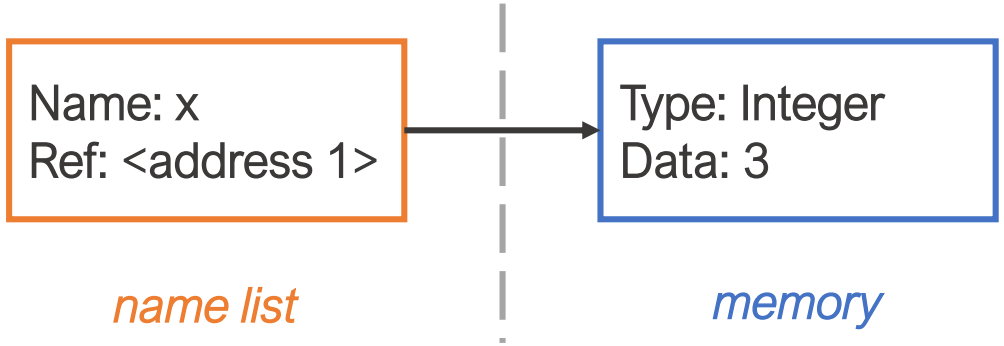
- Зошто?

Семантиката на референците (2)

- Кога се извршува следната наредба се случуваат повеќе работи:

```
x = 3
```

- Прво, се креира цел број (integer) 3 и се зачувува во меморијата
- Се креира променлива *x*
- На променливата *x* се доделува референца до мемориската локација каде се чува *3*
- Значи: Кога кажуваме дека вредноста на *x* е *3*
- Сакаме да кажеме дека *x* сега покажува кон *3*



Семантиката на референците (3)

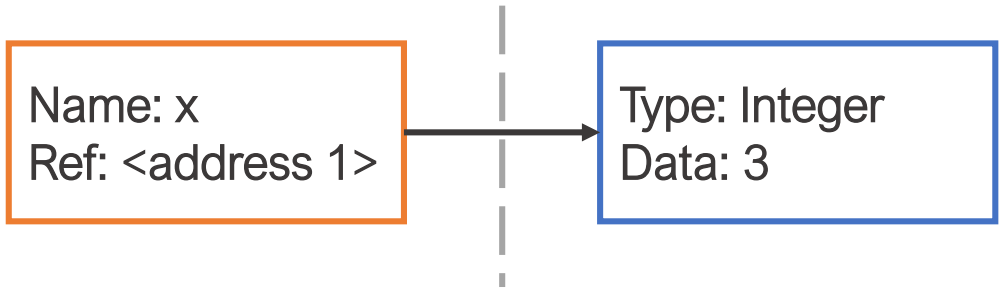
- Податокот 3 што го креиравме е од тип integer. Во Python, типовите integer, float и string (и tuple) се “немутирачки”.
- Тоа не значи дека не можеме да ја промениме вредноста на x, односно да промениме кон што покажува x…
- На пример, може да се зголеми x:

```
>>> x = 3
>>> x = x + 1
>>> print(x)
4
```

Семантиката на референците (4)

- Кога се зголемува x, всушност се случува:
  1. Се наоѓа референцата на променливата *x*.
  2. Се зема вредноста кон која покажува таа референца.

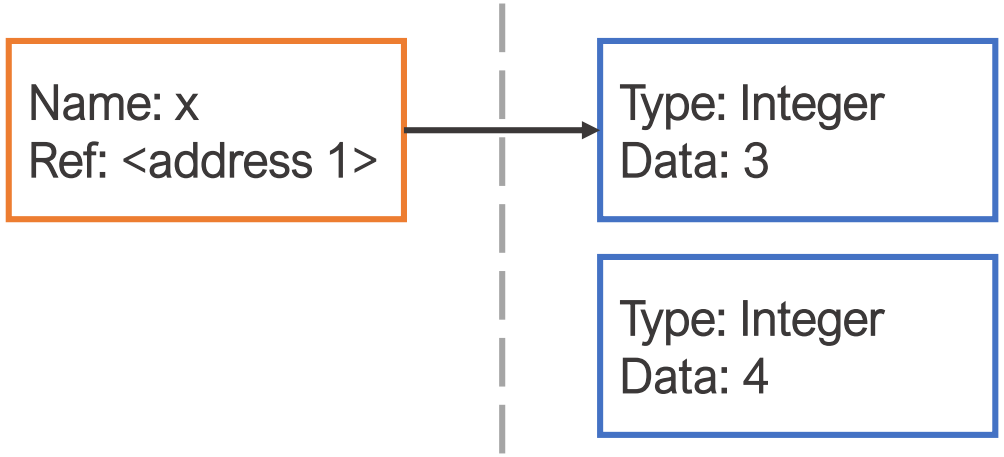
```
>>> x = x + 1
```



Семантиката на референците (5)

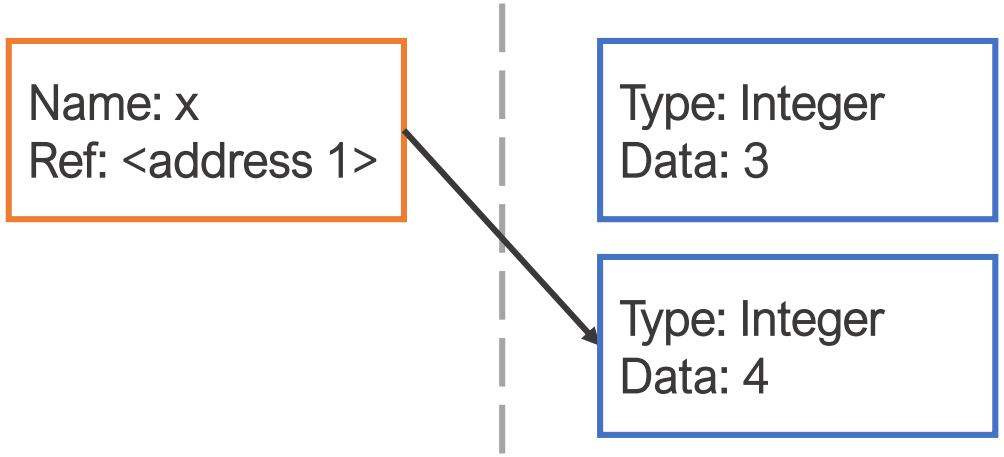
- Кога се зголемува x, всушност се случува:
  1. Се наоѓа референцата на променливата x.
  2. Се зема вредноста кон која покажува таа референца.
  3. Се пресметува 3+1 со што се добива нов податочен елемент *4* кој се чува на нова мемориска локација со нова референца.

```
>>> x = x + 1
```



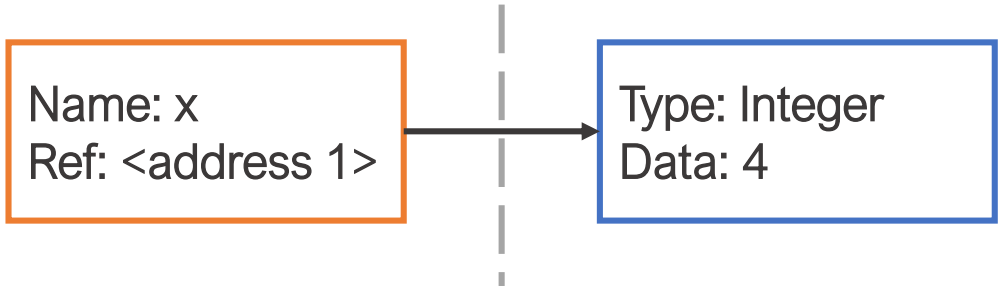
Семантиката на референците (6)

- Кога се зголемува x, всушност се случува:
  1. Се наоѓа референцата на променливата x.
  2. Се зема вредноста кон која покажува таа референца. >>> x = x + 1
  3. Се пресметува 3+1 со што се добива нов податочен елемент 4 кој се чува на нова мемориска локација со нова референца.
  4. Се променува променливата x да покажува кон новата референца (новата мемориска локација).



Семантиката на референците (6)

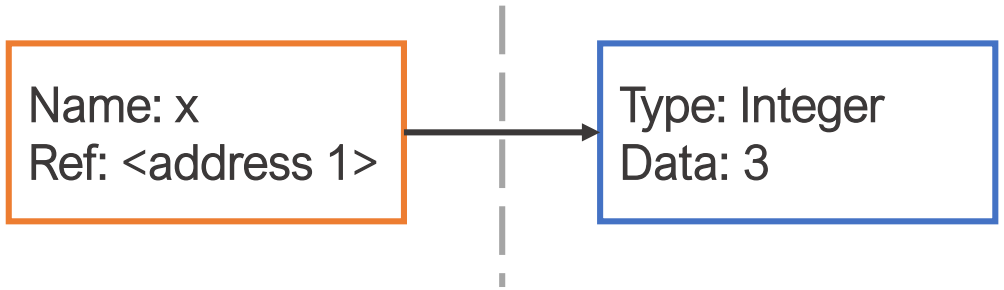
- Кога се зголемува x, всушност се случува:
  1. Се наоѓа референцата на променливата x.
  2. Се зема вредноста кон која покажува таа референца. >>> x = x + 1
  3. Се пресметува 3+1 со што се добива нов податочен елемент 4 кој се чува на нова мемориска локација со нова референца.
  4. Се променува променливата x да покажува кон новата референца (новата мемориска локација).
  5. Garbage collector-от ја брише вредноста 3 ако веќе никоја друга променлива не ја референцира.



Доделување вредности

- Значи за едноставни вградени податочни типови (integer, float, стрингови), доделувањето се однесува како што очекуваме:

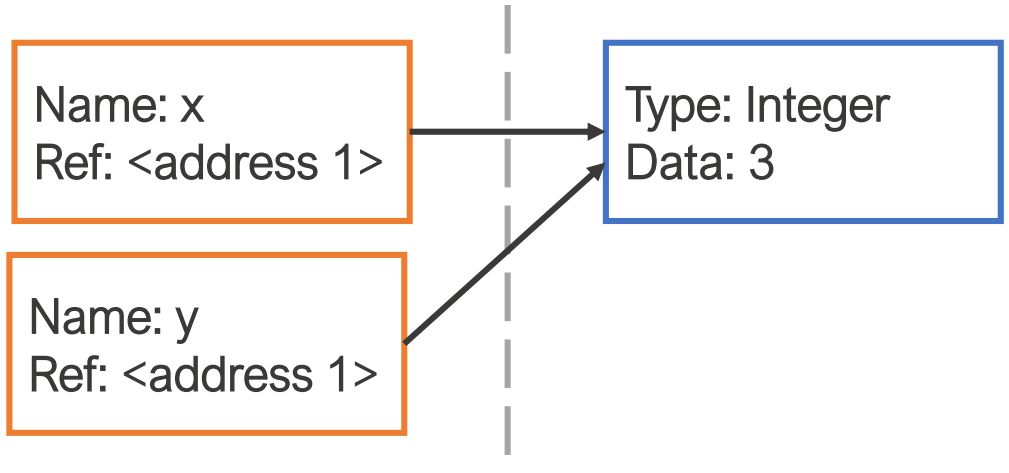
```
>>> x = x + 1          # Creates 3, name x refers to 3.
>>> print(x)           # No effect on x, still ref 3.
3
```



Доделување вредности (2)

- Значи за едноставни вградени податочни типови (integer, float, стрингови), доделувањето се однесува како што очекуваме:

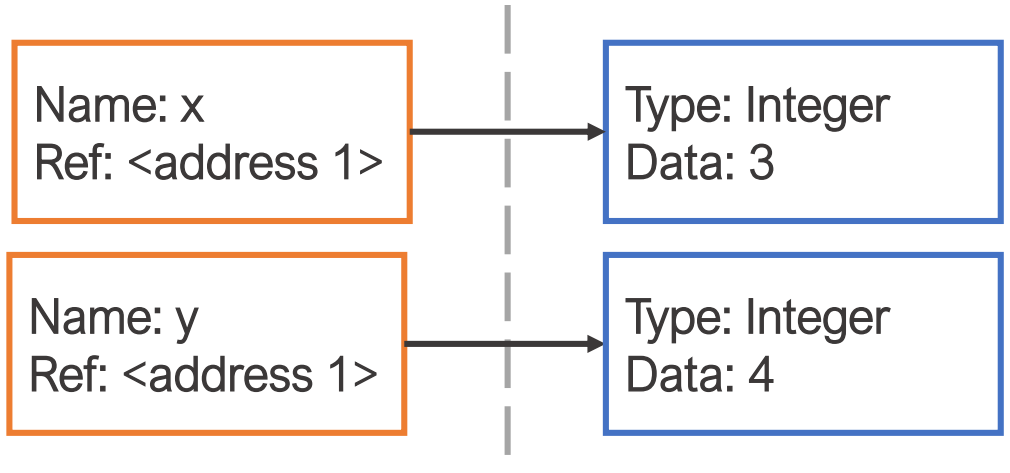
```
>>> x = x + 1          # Creates 3, name x refers to 3.
>>> print(x)           # No effect on x, still ref 3.
3
>>> y = x              # Creates name y, refers to 3.
```



Доделување вредности (3)

- Значи за едноставни вградени податочни типови (integer, float, стрингови), доделувањето се однесува како што очекуваме:

```
>>> x = x + 1          # Creates 3, name x refers to 3.
>>> print(x)           # No effect on x, still ref 3.
3
>>> y = x              # Creates name y, refers to 3.
>>> y = 4              # Creates ref for 4. Changes y.
```



Доделување вредности (4)

- За останатите податочни типови (листи, речници, кориснички дефинирани типови) доделувањето вредности се прави поинаку:
  - Овие податочни типови се **“мутабилни”**.
  - Кога се прави промена на вредностите тоа се прави на истата мемориска локација
  - Не се копираат на нова мемориска локација секој пат
  - Ако напишеме: y=x а потоа го промениме y, и x и y се менуваат!

Immutable

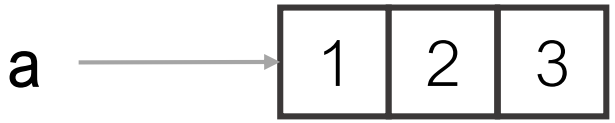
```
>>> x = 3
>>> y = x
>>> y = 4
>>> print(x)
3
```

Mutable

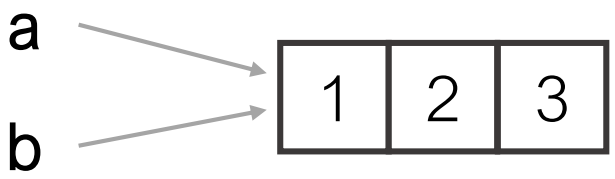
```
x = some mutable object
y = x
make change to y
look at x
X will be changed as well
```

Промена на листа референцирана од 2 променливи

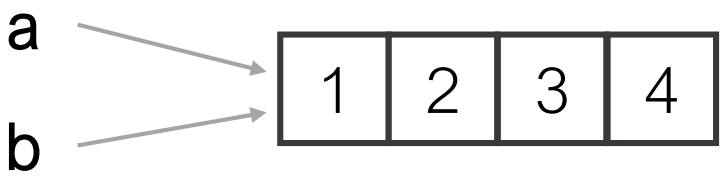
```
a = [1, 2, 3]
```




```
b = a
```




```
a.append(4)
```



Секвенциски типови: Торки (Tuples), Листи (Lists) и Стрингови (Strings)

Tuples = ()
Lists = []
Strings = " "

Секвенциски типови

1. Торка (Tuple)
   - Едноставна *немутирачка* подредена секвенца на елементи (items)
   - Елементите може да бидат од мешани типови, вклучително и секвенциски типови

2. Стринг (String)
   - *Немутирачки*
   - Концептуално се многу слични на торките

3. Листа (List)
   - *Мутирачка* подредена секвенца од елементи од мешани типови

Слична синтакса

- Сите три типови (tuples, strings, lists) имаат слична синтакса и функционалност.

- Главна разлика:
  - Торките и стринговите се *немутабилни*
  - Листите се *мутабилни*

- Операциите на следните слајдови може да се применат на сите секвенциски типови
  - иако (некои од) примерите се однесуваат само на еден од типовите

Секвенциски типови

- Торките (tuples) се дефинираат со употреба на мали загради (и запирки):

```
tu = (23, 'abc', 4.56, (2,3), 'def')
```

- Листите (lists) се дефинираат со квадратни (средни) загради (и запирки):

```
li = ["abc", 34, 4.34, 23]
```

- Стринговите (strings) се дефинираат со наводници (", ‘, или """):

```
st = "Hello"
st = 'Hello'
st = """Ova e povekjeliniski string
        koj koristi trojni nadovnici."""
```

Секвенциски типови (2)

- Може да се пристапуваат поединечните членови на торите, листите и стринговите со помош на квадратни загради (слично како со низите во C/C++).
- *Првите елементи имаат индекс 0…*

```
>>> tu = (23, 'abc', 4.56, (2,3), 'def')
>>> tu[1]                # Second item in the tuple.
'abc'

>>> li = ["abc", 34, 4.34, 23]
>>> li[1]                # Second item in the list.
34

>>> st = "Hello World"
>>> st[1]                # Second item in the string.
'e'
```

Позитивни и негативни индекси

```
          0     1      2      3      4
tu = (23, 'abc', 4.56, (2,3), 'def')
         -5    -4     -3     -2     -1
```

Позитивен индекс: број на елементи од лево, почнувајќи од 0.

```
>>> tu[1]                # Second item from the start.
'abc'
```

Негативен индекс: број на елементи од десно, почнувајќи од –1.

```
>>> tu[-3]               # Third element from the end.
4.56
```

Потсекување: земање на копија од подмножество

```
tu = (23, 'abc', 4.56, (2,3), 'def')
```

Добивање на копија од секвенца со подмножество од оригиналните членови. Почеток на копирањето е првиот индекс, а крај е елементот *пред* вториот индекс.

```
>>> tu[1:4]           # Vtori, treti i cetvrti element.
('abc', 4.56, (2,3))
```

Може да се користат и негативни индекси за потсекување.

```
>>> tu[1:-1]          # Vtori, treti i cetvrti element.
('abc', 4.56, (2,3))
```

Потсекување: земање на копија од подмножество (2)

```
tu = (23, 'abc', 4.56, (2,3), 'def')
```

Се изоставува првиот индекс за да се копира од почетокот до пред елементот чиј индекс е вториот индекс.

```
>>> tu[:2]
(23, 'abc')
```

Се изоставува вториот индекс за да се копира почнувајќи од првиот индекс до крајот на контејнерот.

```
>>> tu[2:]
(4.56, (2,3), 'def')
```

Копирање на целиот контејнер

За да се копира целата секвенца може да се користи [:].

```
>>> tu[:]
(23, 'abc', 4.56, (2,3), 'def')
```

Кај мутирачките секвенци следниве две наредби се различни:

```
>>> list2 = list1          # 2 имиња покажуваат кон референцата 1.
                           # Промена на едната влијае на двете.
>>> list2 = list1[:]       # Две независни копии, две референци.
```

Операторот „in“

- Логичко тестирање за проверка на припадност на одредена вредност во контејнер:

```
>>> t = [1, 2, 4, 5]
>>> 3 in t
False
>>> 4 in t
True
>>> 4 not in t
False
```

- За стрингови, проверка на подстрингови:

```
>>> a = 'abcde'
>>> 'c' in a
True
>>> 'cd' in a
True
>>> 'ac' in a
False
```

- Внимателно: клучниот збор *in* се употребува и за *for циклуси* и за *листи*.

Операторот +

- Операторот + дава *нова* торка, листа или стринг чија вредност е спојување на последните аргументи.

```
>>> (1, 2, 3) + (4, 5, 6)
(1, 2, 3, 4, 5, 6)

>>> [1, 2, 3] + [4, 5, 6]
[1, 2, 3, 4, 5, 6]

>>> "Hello" + " " + "World"
"Hello World"
```

Операторот *

- Операторот * дава *нова* торка, листа или стринг кој го повторува првиот операнд онолку пати колку што е вредноста на вториот операнд.

```
>>> (1, 2, 3) * 3
(1, 2, 3, 1, 2, 3, 1, 2, 3)

>>> [1, 2, 3] * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]

>>> "Hello" * 3
"HelloHelloHello"
```

Мутабилност: Торки наспроти Листи

Tuples = ()

vs.

Lists = []

Торки: Немутабилни

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
>>> t[2] = 3.14
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    t[2] = 3.14
TypeError: 'tuple' object does not support item
assignment
```

Не може да се менуваат торките!!!

Може да се направи нова торка и да се додели на некоја претходна променлива:

```
>>> t = (23, 'abc', 4.56, (2,3), 'def')
>>> t1 = (23, 'abc', 3.14, (2,3), 'def')
>>> t = t1
```

Листи: Мутабилни

```
>>> li = ['abc', 23, 4.34, 23]
>>> li[1] = 45
>>> li
['abc', 45, 4.34, 23]
```

- Листите може да се менуваат директно.
- Променливата *li* сè уште покажува на истата мемориска локација.
- Мутабилноста на листите има цена и затоа се побавни од торките.

Операции само за листи

```
>>> li = [1, 2, 3, 4, 5]

>>> li.append('a')
>>> li
[1, 2, 3, 4, 5, 'a']

>>> li.insert(2, 'i')
>>> li
[1, 2, 'i', 3, 4, 5, 'a']
```

Методот extend и операторот +.

- + креира нова листа (со нова меморискa локација)
- extend оперира на листата li на истата меморискa локација (in place).

```
>>> li = [1, 2, 3] + [4, 5, 6]
>>> li.extend([9, 8, 7])
>>> li
[1, 2, 3, 4, 5, 6, 9, 8, 7]
```

Збунувачки:

- Extend прима листа како аргумент.
- Append прима еден елемент како аргумент, па ако се проследи листа целата листа се третира како еден елемент.

```
>>> li = [1, 2, 3] + [4, 5, 6]
>>> li.append([10, 11, 12])
>>> li
[1, 2, 3, 4, 5, 6, 9, 8, 7, [10, 11, 12]]
```

Операции само за листи (2)

```
>>> li = ['a', 'b', 'c', 'b']

>>> li.index('b')              # index of the first occurrence
1

>>> li.count('b')              # number of occurrences
2

>>> li.remove('b')             # remove first occurrence
>>> li
['a', 'c', 'b']
```

Операции само за листи (3)

```
>>> li = [5, 2, 6, 8]

>>> li.reverse()                # reverse the list *in place*
>>> li
[8, 6, 2, 5]

>>> li.sort()                   # sort the list *in place*
>>> li
[2, 5, 6, 8]

>>> li.sort(some_function)      # sort in place using user-defined
comparison
```

Торки наспроти Листи

- Листите се побавни, но помоќни од торките.
  - Листите може да се менуваат и постојат повеќе операции кои може да се извршуваат над нив.
  - Торките се немутирачки, но побрзи и поддржуваат помалку операции.
- Може да се направи конверзија од еден во друг тип со функциите list() и tuple():

```
li = list(tu)

tu = tuple(li)
```

Речници (Dictionaries)

Dict = {}

Речници (Dictionaries): Мапирачки тип

- Речниците чуваат мапирање на множество од клучеви и множество од вредности.
  - Клучевите може да бидат било кој немутирачки тип.
  - Вредностите може да бидат од било кој тип.
  - Еден речник може да чува вредности од различни типови.
- Може да се дефинираат, менуваат, гледаат, пребаруваат и бришат паровите од клучеви и вредности (key-value pairs) во речникот.

Употреба на речници (dictionaries)

```
>>> d = {'user': 'bozo', 'pswd': 1234}
>>> d['user']
'bozo'
>>> d['pswd']
1234
>>> d['bozo']
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    d['bozo']
KeyError: 'bozo'
```

```
>>> d = {'user': 'bozo', 'pswd': 1234}
>>> d['user'] = 'clown'
>>> d
{'user': 'clown', 'pswd': 1234}
>>> d['id'] = 45
{'user': 'clown', 'id': 45, 'pswd': 1234}
```

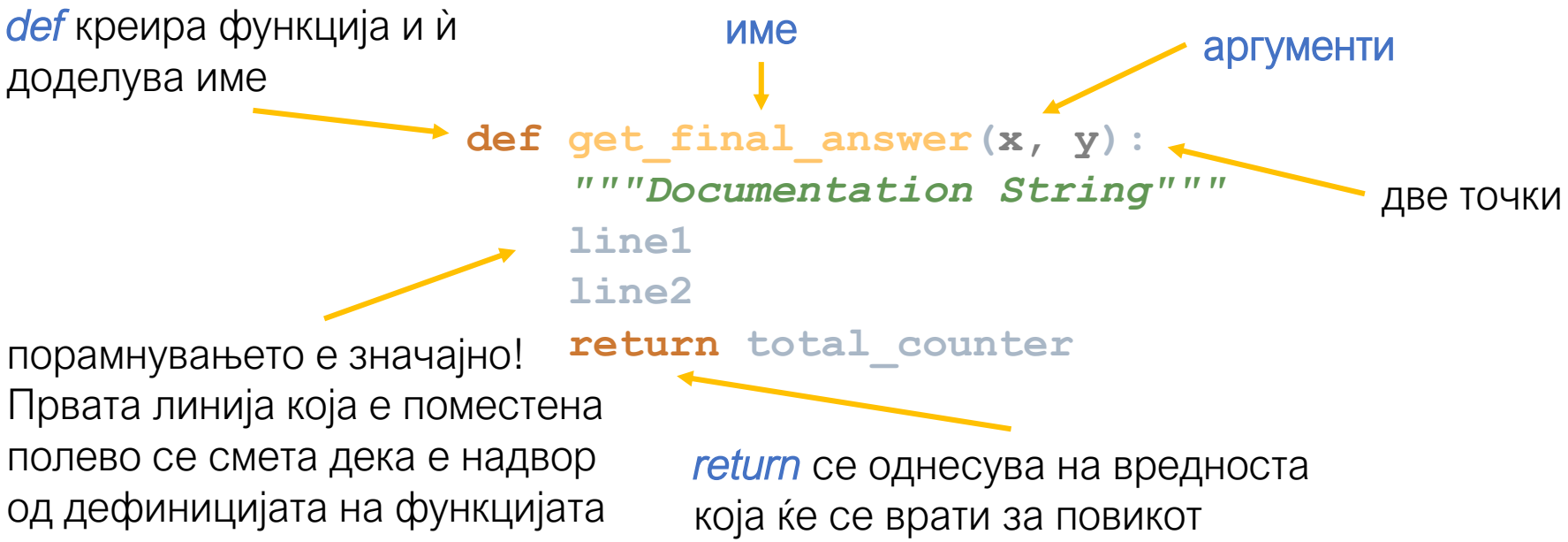
```
>>> d = {'user': 'bozo', 'p': 1234, 'i': 34}
>>> del d['user']       # remove one
>>> d
{'p': 1234, 'i': 34}
>>> d.clear()           # remove all
>>> d
{}
```

```
>>> d = {'user': 'bozo', 'p': 1234, 'i': 34}
>>> d.keys()            # list of keys
['user', 'p', 'i']
>>> d.values()          # list of keys
['bozo', 1234, 34]
>>> d.items()           # list of item tuples
[('user', 'bozo'), ('p', 1234), ('i', 34)]
```

Функции во Python

```
def func()
```

Дефинирање на функции



Функциите не се дефинираат во header-от ниту пак се декларира типот на функцијата и нејзините аргументи

Повик на функција

- Синтакса за повик на функции:

```
>>> def my_fun(x, y):
        return x * y
>>> my_fun(3, 4)
12
```

- Доделувањето на вредности на параметрите на функциите го следи истото правило како било која друга променлива во Python
  - семантика на референци

Доделување вредности

- *Сите* функции во Python враќаат вредност
  – дури и кога *return* линијата е изоставена од дефиницијата
- Функциите без *return* враќаат специјална вредност *None*.
  – *None* е специјална константа во Python.
  – *None* се користи како *NULL*, *void*, или *nil* во другите програмски јазици.
  – *None* е логички еквивалент на False.
  – Интерпретерот не ја прикажува *None* вредноста.

Преоптоварување на функции? НЕ.

- Во Python функциите не може да се преоптоварат.
  - За разлика од C++, функциите во Python се специфицираат по нивното име
    - Бројот, редоследот, имињата, и типовите на аргументи не можат да се искористат за диференцирање помеѓу две функции со исто име.
    - Две функции не можат да имаат исто име, иако имаат различни аргументи.

Функциите се објекти од прв ред во Python

- Функциите можат да се користат како било кој друг податочен тип
- Една функција може
  - да биде аргумент на друга функција
  - да биде return вредност во дефиниција на функција
  - да се додели на некоја променлива
  - да биде дел од торка, листа, итн.
  - ...

```
>>> def my_fun(x):
        return x * 3
>>> def applier(q, x):
        return q(x)
>>> applier(my_fun, 7)
21
```

Специфична синтакса за функции

```
def func()
```

Lambda нотација

- Во Python може да се дефинираат функции и без да бидат експлицитно именувани
  – анонимни или неименувани функции
- Оваа можност најмногу се користи кога сакаме некоја кратка функција да ја проследиме како аргумент на друга функција

```
>>> applier(lambda z: z * 4, 7)
28
```

- Првиот аргумент на applier() е неименувана функција која има една променлива на влез и како резултат ја враќа вредноста на таа променлива помножена со 4
- Забелешка: со помош на lambda нотацијата можат да се дефинираат само неименувани функции кои излезот го генерираат со еден единствен израз

Подразбрани вредности за аргументи

- Можете да дефинирате подразбирани вредности за аргументите на вашите функции
- Вака дефинираните аргументи се опционални при повик на функцијата.

```
>>> def my_fun(b, c=3, d="Hello"):
        return b + c

>>> my_func(5, 3, "hello")
>>> my_func(5, 3)
>>> my_func(5)
```

Сите прикажани повици враќаат резултат 8

Редослед на аргументи

- Функција може да се повика со променет редослед на дел од аргументите (или сите аргументи) под услов да ги специфицирате аргументите при самиот повик

```
>>> def my_fun(a, b, c):
        return a - b
>>> my_func(2, 1, 43)
1
>>> my_func(c=43, b=1, a=2)
1
>>> my_func(2, c=43, b=1)
1
```