



Преговор на основни понятия. Програма,  
променливи, константи.  
Основни понятия в езика Python.  
Примерни програми.

Структури от данни и програмиране - преговор



# Алгоритми. Език за програмиране.

# Алгоритъм



Крайна последователност от  
елементарни за изпълнителя  
инструкции, водеща до определен  
резултат



*Абу Джафар Мухаммед ибн Муса ал-Хорезми*

Входни данни →

Алгоритъм

→ Изходни данни

# Свойства на алгоритмите



- **Масовост**  
Може да се прилага за коя да е задача от клас еднотипни задачи
- **Дискретност**  
Състои се от последователни, различни една от други стъпки
- **Детерминираност (определеност)**  
При всяко изпълнение с едни и същи входни данни се получава един и същ резултат
- **Изпълнимост**  
Всяка от използваните операции е съобразена с възможностите на изпълнителя
- **Крайност**  
Състои се от краен брой стъпки. Завършва работа за крайно време.
- **Сложност (ефективност)**  
Оценка на използваните ресурси (памет или време) в зависимост от размера на входните данни

# Пример – Фалшивата монета



Разполагате с  $n$  златни монети и везна без деления и теглилки. Една от монетите е фалшива – по-лека от останалите.

С колко най-малко претегления може да се открие фалшивата монета?

Как?



# Видове алгоритми



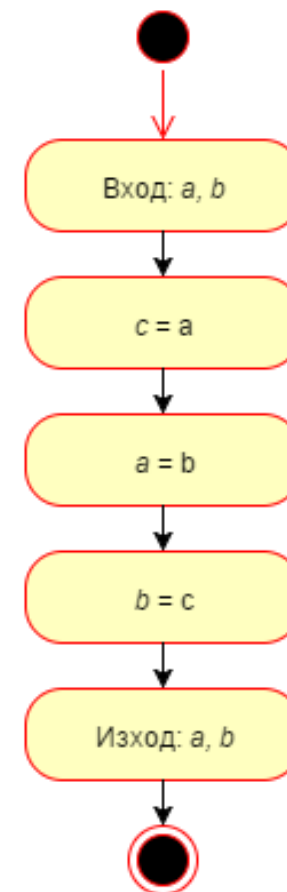
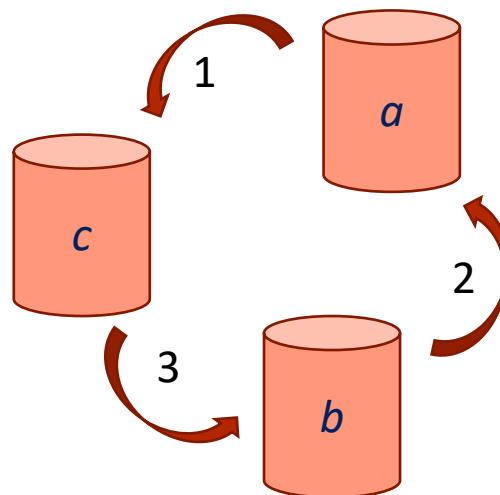
- Линејни
- Разклонени
- Циклични

# Линейни алгоритми



Размяна на стойности на две променливи

- Безусловна последователност от инструкции
- Пример: размяна на стойности на две променливи

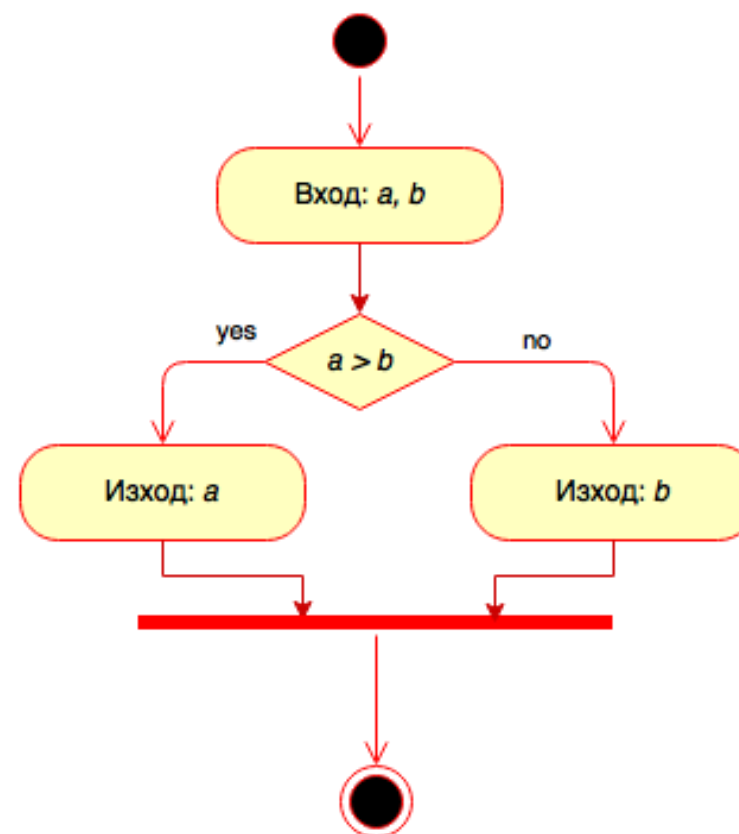


# Разклонени алгоритми



- Потокът от инструкции зависи от едно или повече условия
- Пример: намиране на по-голямото от две числа

Намиране на максимален  
елемент



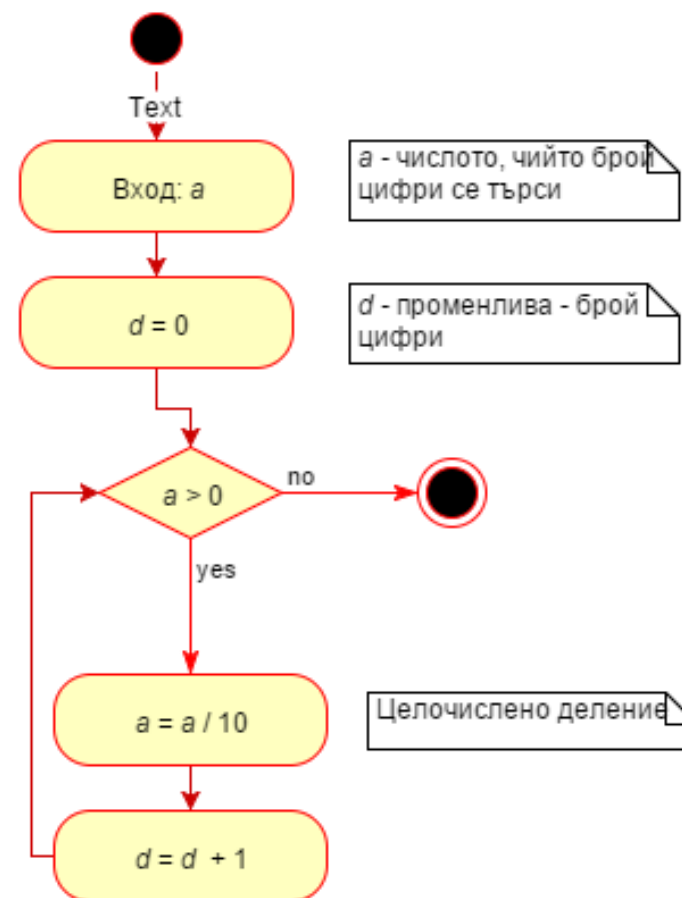


# Циклични алгоритми



- Потокът от инструкции съдържа повтарящи се групи
- Пример: намиране на броя на цифрите на число, записано в десетична бройна система

Намиране на броя на цифрите на число, записано в 10-ична бройна система



# Методи за описание на алгоритми

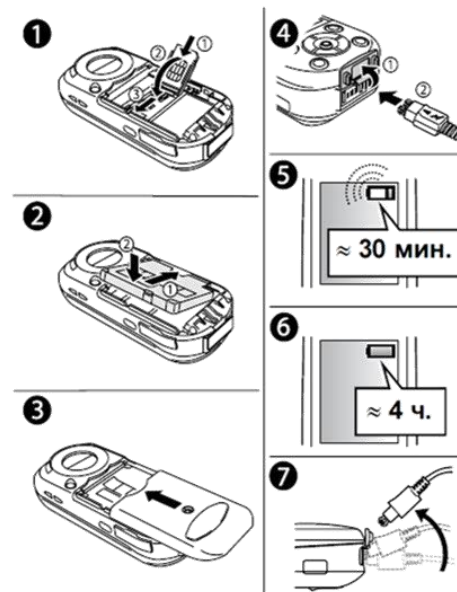



Начало на работата

- Словесен  
Пример – готварска рецепта
- Графичен  
Примери – последователност от изображения, диаграма
- Чрез формален език  
Пример – език за програмиране

Предимства и недостатъци на всеки метод?

**За поставяне на SIM картата и зареждане на батерията:**



- 1 Повдигнете държача на SIM картата в посоката на отваряне, отбелязана с OPEN. Плъзнете SIM картата в прорезите на плъзгача, като златните контакти сочат надолу.
- 2 Поставете батерията откъм гърба на телефона, с етикета нагоре и така, че съединителите да са един към друг.
- 3 Поставете капачето на батерията, както е показано на картината, и го плъзнете на място.
- 4 Отворете капачето на съединителя и свържете зарядното устройство към телефона при символа на светкавица. Символът на светкавица върху щепсела на зарядното устройство трябва да е отгоре.
- 5 При зареждане, докато на екрана се появи иконата на батерията, може да изминат до 30 минути.
- 6 Заредете я в продължение на 4 часа или докато иконата на батерията покаже пълно зареждане. Ако след този период не виждате иконата на батерия, натиснете произволен клавиш или , за да активирате екрана.
- 7 Извадете зарядното устройство, като извийте щепсела нагоре.

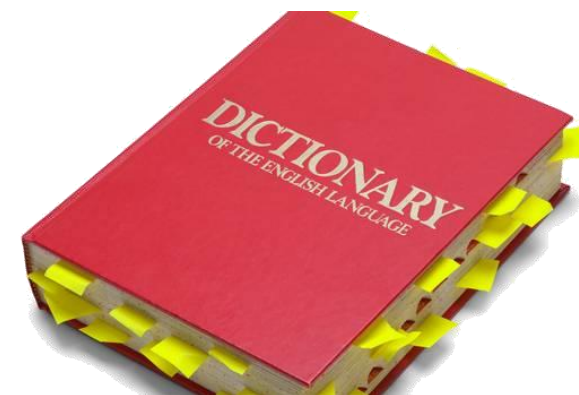


# Език и среда за програмирање

# Език за програмиране



- Език за програмиране – изкуствен, формализиран език
  - Характеристики:
    - Речник
    - Граматика
  - Речник
    - Запазени думи
    - Стандартни / вградени думи
    - Потребителски дефинирани думи
  - Граматика
    - Синтаксис – структура, правила за описание на конструкциите
    - Семантика – смисъл на конструкциите, начинът, по който компютърът го „разбира“
- 
- Какви запазени думи в езика Python познавате?
  - Какви граматически конструкции в езика Python познавате?



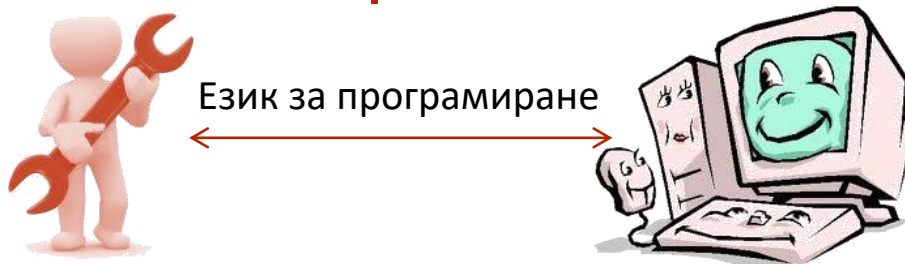
Let's eat grandpa.  
Let's eat, grandpa.

**correct punctuation can  
save a person`s life.**

# Компютърна програма



Недвусмислена, подредена последователност от изчислителни инструкции, необходими за решаването на даден проблем от компютър

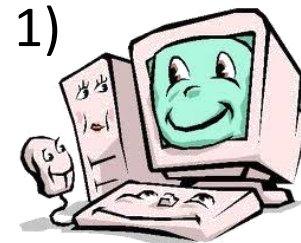


**Високо ниво:** по-близък до естествен за човека (английски, български)

- C++, C#
- Python
- Basic...

**Ниско ниво:** по-близък до езика на компютъра (0 и 1)

- Байт код
- Машинен език
- Асемблер ...



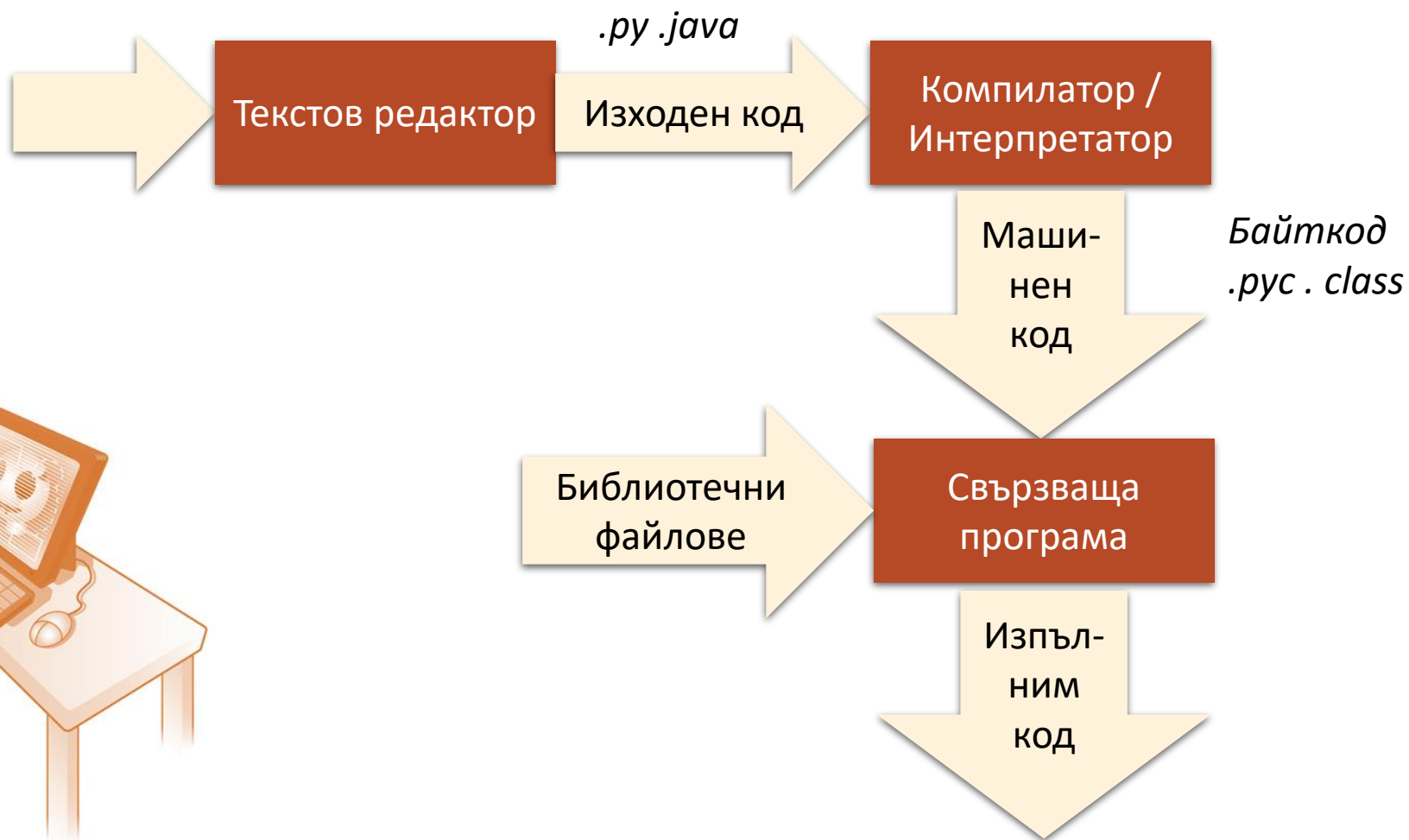
# Транслатори



Специализирани програми, превеждащи програмен код от език за програмиране на машинно изпълним език

- Компилатори - превеждат цялата програма
- Интерпретатори – превеждат единична команда от програма

# Етапи от създаването на компютърна програма



# Програми на PYTHON



- **Всяка програма** съдържа модули
- **Всеки модул** съдържа дефиниции на класове и функции и изпълними команди
- **Командите** съдържат изрази
- **Изразите** съдържат обекти и оператори (обръщения към функции) и се изчисляват (оценяват) до получаване на стойност (обект)



# Приоритет на операторите в изразите



Операторите подредени в низходящ ред по приоритет:

- 1.  $-x$ ,  $+x$ ,  $\sim x$ ,  $**$
- 2.  $*$ ,  $\%$ ,  $/$ ,  $//$
- 3.  $+$ ,  $-$
- 4.  $<<$ ,  $>>$
- 5.  $\&$
- 6.  $\wedge$
- 7.  $|$
- 8.  $=$ ,  $+=$ ,  $-+$ ,  $*=$ ,  $/=$ ,  $//=$ ,  $\%=$ ,  $**=$

# Приоритет на операторите в изразите



- В един израз поредността на прилагане на операторите се определя от наличието на скоби
- При липса на скоби, операторите се прилагат в ред от най-високия приоритет към най-малкия:
- $3+4*5 = 3 + (4 * 5) = 23$
- При последователност от оператори с еднакъв приоритет, те се изпълняват последователно от ляво на дясно

# Вградени типове данни (класове)



Типове	Примери за стойности
Числа – int, float, complex	1234, 3.1415, 3+4j
Низове – str, bytes, bytearray	'spam', "guido's"
Логически - bool	True, False
Списъци- list	[1, [2, 'three'], 4]
Речници - dict	{'food': 'spam', 'taste': 'yum'}
Редици - tuple	(1,'spam', 4, 'U')
Множества - set	{1, 3, 25, 6, 4, 18}

# Обекти в PYTHON



- **Обектите** са данните в програмите
- **Всеки обект** има идентичност (адрес в ОП където се съхранява), тип (клас) и стойност
- **Идентичността** на един обект не се променя
- **Типът** на един обект не се променя
- **Стойността** на някои обекти може да се променя (**mutable**), а на други не може да се променя (**immutable**)
- **Променливите** се използват за именуване на обектите

# Mutable / Immutable



- **Mutable:** списъци, речници (стойностите), множества
- **Immutable:** числа, низове, логически, редици, ключовете в речниците
- **Контейнери:** списъци, редици, речници, множества
- Във всеки **Immutable контейнер**, ако се съдържа елемент от тип **mutable**, той може да си променя стойността

# Променливи



- Не се декларират (описват) - направо се използват в команда за присвояване
- Можем да проверим за стойност на променлива, като зададем името и. Това предизвиква извеждане на стойността (защото променливата е израз)
- Променливите трябва да са създадени преди да се използват
- При грешки Python използва механизъм на грешки и изключения

# Присвояване



- Създаване на променлива в Python означава да присвоим на **име** стойност (**адрес** в ОП където се съхранява **обект** с **определен тип и стойност**)
  - Присвояването създава връзка а не нов обект
- Имената в Python нямат тип, обектите имат тип
  - Python определя типа на обекта към който има връзка автоматично в зависимост от стойността съхранявана там
- Име се свързва с обект в момента в който се появява в лявата част на команда за присвояване, а обектът е в дясната част (или се изчислява в дясната част):  
**x = 3**
- Всеки обект и връзка се изтриват ако вече няма променлива която да е свързана с обекта

# Присвояване



- За няколко променливи

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

```
>>> x
```

2

```
>>> y
```

3

Размяна на стойности:

```
>>> x, y = y, x
```

- Верижно присвояване

```
>>> a = b = x = 2
```



# Разширено многозначно присвояване



```
>>> m, *n = 3, 4, 5, 6
```

```
>>> m, n
```

```
(3, [4, 5, 6])
```

```
>>> *m, n = (2, 4, 1, 3, 8, 5)
```

```
>>> m, n
```

```
([2, 4, 1, 3, 8], 5)
```

```
>>> a, *b, c = [1, 2, 3, 4, 5]
```

```
>>> a, b, c
```

```
(1, [2, 3, 4], 5)
```

# Разширено многозначно присвояване



- \* се съпоставя с 0 / произволен брой
- прилага се последна при съпоставяне
- \* връща **СПИСЪК** с елементи
- прилага се за произволен тип контейнер, като връща резултат списък с елементи от контейнера

# Проверка за равенство



Проверка за равенство се прави с двойно = (==)

- При проверка за равенство може да се направи преобразуване на типа
- Идентичност се проверява с оператор `is`

```
>>> 1==1
```

```
True
```

```
>>> 1.0==1
```

```
True
```

```
>>> "1"==1
```

```
False
```

# Арифметични оператори



```
>>> a = 10
>>> a = 10    # 10
>>> a += 1    # 11
>>> a -= 1    # 10
>>> b = a + 1  # 11
>>> c = a - 1  # 9
>>> d = a * 2  # 20
>>> e = a / 2   # 5
>>> f = a % 3   # 1
>>> g = a // 3  # 3
>>> h = a ** 2  # 100
```

# Арифметични сравнения



```
>>> 5 > 3
```

```
True
```

```
>>> 3 >= 5
```

```
False
```

```
>>> 3 < 5
```

```
True
```

```
>>> 3 <= 5
```

```
True
```

```
>>> 3 == 5
```

```
False
```

```
>>> 3 != 5
```

```
True
```

# Оператори за низове



```
>>> animals = "Cats " + "Dogs "  
>>> animals += "Rabbits"  
>>> print(animals)  
Cats Dogs Rabbits  
>>> fruit = ', '.join(['Apple', 'Banana', 'Orange'])  
>>> print(fruit)  
Apple, Banana, Orange  
>>> date = '%s %d %d' % ('Feb', 20, 2018)  
>>> print(date)  
Feb 20 2018  
>>> name = '%(first)s %(last)s' % {'first': 'Apple',  
'last': 'Microsoft'}  
>>> print(name)  
Apple Microsoft
```

# Оператори за низове



- Конкатениране (слепване)
  - `word = 'help' + x`
  - `word = 'help' 'a'`
- Поднизове
  - `'hello'[2] → 'l'`
  - Парче (slice): `'hello'[1:2] → 'e'`
  - `word[-1] → последен символ`
  - `len(word) → дължина на низ`
  - `immutable`: не може да се променя стойност на елемент в низ.

# Логически тип данни



- Те са резултат от сравнения с оператори и от проверки за идентичност и принадлежност
- Резултатът от всяко сравнение или проверка винаги е една от двете логически константи **True** или **False**
- Същият е резултатът от основните логически оператори **and ; or ; not ; in ...**
- Оценяването на израз с логически оператори спира веднага след като резултатът стане ясен (“short circuit”) – например до първи операнд със стойност **True** при оператор **or** или първи операнд **False** при оператор **and**



# Логически оператори



```
>>> a = True
>>> b = False
>>> a and b
False
>>> a or b
True
>>> not b
True
>>> a and not (b or c)
False
>>> True & 1
1
```

# Списъци



- Могат да имат елементи от различен тип данни
  - `a = ['spam', 'eggs', 100, 1234, 2*2]`
- Има достъп до всеки елемент или под-списък:
  - `a[0] → spam`
  - `a[:2] → ['spam', 'eggs']`
  - `a[::-1] → [4, 1234, 100, 'eggs', 'spam']`
- Списъците могат да се променят (за разлика от низовете)
  - `a[2] = a[2] + 23`
  - `a[2:] = [123, 1234, 4]`
  - `a[0:0] = []`
  - `len(a) → 5`

# Под-Списъци (slicing)



- $a[\text{<начало}=0 \text{ >: <край}=-1 \text{ > } [:\text{<стъпка}=1 \text{ >}]]$
- Връща под-списък на дадения който включва всички елементи на оригиналния от началото до края с изместване в индекса зададено от стъпка.
- Стъпката е незадължителен, по подразбиране е 1
- Когато няма зададена стойност за начало, край или стъпка се ползва тази по подразбиране

# Вградени методи-функции за списъци



- `L.append(4)` # добавя елемент в края
- `L.extend([5,6,7])` # добавя елементи в края
- `L.insert(i, X)` # вмъква елемент в позиция i
- `L.index(X)` # връща позиция на първото  
# срещане на X в L
- `L.count(X)` # връща броят на срещанията  
# на X в L

# Вградени методи-функции за списъци 2



- `L.sort()` # сортиране
- `L.reverse()` # обръщане
- `L.copy()` # копиране
- `L.clear()` # изтрива всички елементи
- `L.pop(i)` # изтрива елемент `i` и го връща  
# като стойност (прем.: -1)
- `L.remove(X)` # изтрива първото срещане на  
# `X` в `L`

# Речници



- Речниците са не подредено множество от обекти
- Обектите могат да бъдат от произволен тип
- Броят на обектите е неопределен и се изменя
- Всеки обект може да бъде произволен контейнер
- Всеки обект в речника се идентифицира с ключ (име)
- Достъпът до обектите става чрез ключа (името)
- Речниците са mutable (като списъците)
- Ключът е immutable може да има само една стойност
- Различните ключове трябва да са с различна стойност

# Речници – прости команди



- `E = {}` # празен речник
- `D = {'име': 'Bob', 'age': 40}` # речник с 2 елемента
- `F = {'Num': 11, 'boss': {'name': 'Bob', 'age': 40}}`
- # Вложен речник в речник
- `D['име'] -> 'Bob'`
- `F['boss']['age'] -> 40`
- `>>> 'age' in D`
- `True`
- `>>> 40 in D`
- `False`

# Речници – вградени методи - функции



- `D.keys()` # връща стойностите на ключовете
- `D.values()` # връща стойностите на обектите
- `D.items()` # връща двойки от стойности
- `D.copy()` # връща копие на речника
- `D.clear()` # изтрива елементите на речника
- `D.update(D2)` # добавя в D елементите от D2
- `D.get(key, default?)` # връща обекта за ключа
- `D.pop(key, default?)` # връща и изтрива
- `D.setdefault(key, default?)` # вмъква ключ
- `D.popitem()` # връща и изтрива случайна 2-ка



# Структура от данни: множество



- Отговаря на математическото понятие множество
- Константи от този тип се задават в  $\{ \}$  разделени със запетайка
- Разграничават се от речник по това, че елементите на речник са двойка ключ:стойност
- Елементите в множеството не са подредени и не се допуска повторения

# Особености на множествата в Python



- Всеки обект от тип множество е `mutable`
- Елементите на всяко множество трябва да бъдат обекти от тип `immutable`
- Това означава, че не може да имаме множество като елемент на друго множество – проблем!
- За това се въвежда специален тип данни: `immutable` множество – `frozenset`, обектите от който вече могат да са елементи на множество

Operation	Math Notation	Python Syntax	Result Type	Meaning
Union	$A \cup B$	<code>A   B</code>	set	Elements in A or B or both
Intersection	$A \cap B$	<code>A &amp; B</code>	set	Elements common to both A and B
Set Difference	$A - B$	<code>A - B</code>	set	Elements in A but not in B
Symmetric Difference	$A \oplus B$	<code>A ^ B</code>	set	Elements in A or B, but not both

Operation	Math Notation	Python Syntax	Result Type	Meaning
Set Membership	$x \in A$	<code>x in A</code>	bool	x is a member of A
Set Membership	$x \notin A$	<code>x not in A</code>	bool	x is not a member of A
Set Equality	$A = B$	<code>A == B</code>	bool	Sets A and B contain exactly the same elements
Subset	$A \subseteq B$	<code>A &lt;= B</code>	bool	Every element in set A also is a member of set B
Proper Subset	$A \subset B$	<code>A &lt; B</code>	bool	A is a subset B, but B contains at least one element not in A

# Редици (tuples)



- Могат да имат елементи от различен тип данни
  - `a = ('spam', 'eggs', 100, 1234, 2*2)`
  - `b = 2018, "Year", 3`
- Наподобяват списъци, но се извеждат в кръгли скоби
- Има достъп до всеки елемент или под-редица:
  - `a[0] → spam`
  - `a[:2] → ['spam', 'eggs']`
- Начинът на индексирание и достъп до елементи или под-редици е същият както при низовете и списъците
- Редиците не могат да се променят (както низовете, и за разлика от списъците)

# Редици 2



- Могат да бъдат с произволна но фиксирана дължина, която може да се разбере с вградената функция `len()`
- Всеки елемент на редицата може да бъде обект от произволен тип данни, включително контейнер, и в частност редица (наричаме го под-редица)
- Всеки обект от тип редица (tuple) е `immutable` – затова дължината на всеки обект не може да се мени
- Всяка редица се съхранява в паметта като многомерен масив от адреси на съответните елементи – обекти (както списъците)

# Оператори за редици



- Допустими са същите както за низове и списъци, имат на практика същото действие:
  - + - слепване (конкатениране)
  - \* - размножаване (копиране)
  - in - проверка за наличие на елемент
- Примери:
- **>>> (1, 2, 3) + (4, 5, 6)**
- (1, 2, 3, 4, 5, 6)
- **>>> ('Hi', ', ', ') \* 3**
- ('Hi', ', ', ', 'Hi', ', ', ', 'Hi', ', ', ')

# Условна команда if



Общ синтаксис на командата:

if <израз 1>:

    <блок команди 1>

elif <израз 2>: # 0 или повече elif клаузи

    <блок команди 2>

else: # Допуска се 0 или 1 клауза else

    <блок команди 3>

<следващи команди ...>



# Условни команди



```
>>> if grade >= 90:
    if grade == 100:
        print('A+')
    else:
        print('A')
elif grade >= 80:
    print('B')
elif grade >= 70:
    print('C')
else:
    print('F')
```

# Цикъл: while



while <test>: # Проверка на клауза <test>  
    <block>       # Ако true: изпълнява блока с команди  
else:            # Незадължителна клауза else  
    <block>       # Изпълнява се след нормален край

# Цикъл: for



For <var> in <object>:

*# Присвоява на променлива*

*# var елементи от <object>*

*<block>*

*# Докато има елементи за присвояване:*

*# изпълнява блока с команди*

else: *# Незадължителна клауза else*

*<block> # Изпълнява се след нормален край*

# Вградени команди за цикли



- ✓ break — моментален изход от текущия цикъл
- ✓ continue — моментален скок към началната клауза на цикъла (пропускат се другите команди в блока)
- ✓ pass — команда която не прави нищо
- ✓ клауза else — изпълнява се само ако цикъла завърши нормално (без команда break)

# Заключение



- Алгоритми
- Компютърна програма
- Среда за програмиране
- Основни елементи в програма на Python