

Основи на Програмирането

Лекция 3

Вградени типове данни в езика Python
Числови, логически и символни данни



Какво ще научите

- ✓ Идентичност на обекти
- ✓ Цели, реални и комплексни числа
- ✓ Вградени функции за работа с числа
- ✓ Какво представляват низовете
- ✓ Как се представят и кодират символите във файловете
- ✓ Какво е Escape последователност
- ✓ Видове низове
- ✓ Вградени оператори и команди
- ✓ Вградени методи и функции

Програми на PYTHON

- **Всяка програма съдържа модули**
- **Всеки модул съдържа команди**
- **Командите съдържат изрази**
- **Изразите създават и обработват обекти**

Обекти в PYTHON

- **Обектите** са абстрактното представяне на данните в програмите
- **Всеки обект** има идентичност (адрес в ОП където се съхранява), тип 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**)
- Всеки обект има **брояч на използване**
- Всеки обект има **подразбираща се логическа стойност** (дали `bool(<обект>)` е `True` или `False`)

Идентичност на обекти в PYTHON

- Два обекта се сравняват дали имат една и съща идентичност с командата *is*
- Идентичността на един обект се получава с функцията *id()*

```
>>> 1 is 2
```

```
False
```

```
>>> 1 is 1
```

```
True
```

```
>>> id(1)
```

```
506386224
```

Идентичност на обекти в PYTHON

Друг пример:

```
>>> a=5
```

```
>>> id(a)
```

```
140438614885096
```

```
>>> a=10
```

```
>>> id(a)
```

```
140438614884976
```

```
>>> b=a
```

```
>>> id(b)
```

```
140438614884976
```

Тип на обекти в PYTHON

- **Типът на обектите** определя операциите с тях (събиране или слепване или дължина) и стойностите, които могат да приемат
- **Типът на един обект** се проверява с функцията *type()*

```
>>> type(1)
```

```
<class 'int'>
```

```
>>> type([])
```

```
<class 'list'>
```

Контейнери

- Това са обекти които имат като елементи указатели (идентичност) на други обекти

Пример: [1, a, “help”, True]

- Стойността на един контейнер включва стойностите на отделните му елементи
- Ако контейнерът е **immutable**, това означава че указателите на елементите не се променят. Но ако даден елемент е от тип **mutable**, стойността му може да се промени.

Вградени типове

Типове	Примери за стойности
Числа	1234, 3.1415, 3+4j
Низове	'spam', "guido's"
Логически	True, False
Списъци	[1, [2, 'three'], 4]
Речници	{'food': 'spam', 'taste': 'yum'}
Редици	(1,'spam', 4, 'U')
Файлове	myfile = open('eggs', 'r')

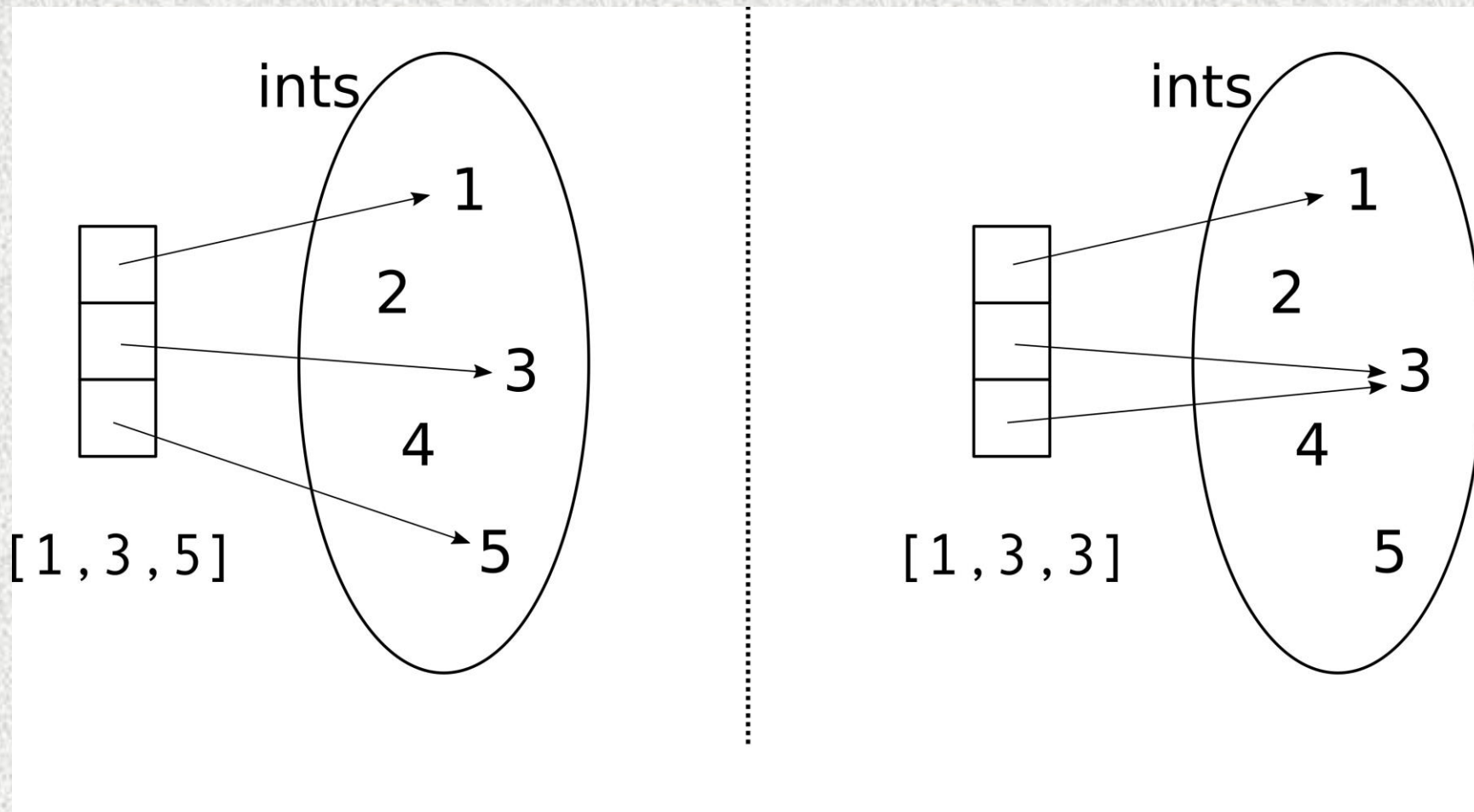
Други вградени типове

- ✓ **None** – този тип има само една стойност: **None** (това е и запазена дума). Използва се за обекти без стойност. Логическата му еквивалентна стойност е **False**
- ✓ **NotImplemented** – подобен на горния, но се използва за методи (функции), които не могат да се приложат за аргументите, с които са извикани. Логическата му еквивалентна стойност е **True**.
- ✓ **Ellipsis (...)** – задава се в дефиниции на класове и функции когато не знаем какви команди да сложим, както и за задаване на много мерни индекси. Логическата му стойност е **True**

Представяне на обектите

- ✓ Обектите от прости (не съставни) типове данни се представят в ОП като поредица от битове
- ✓ Те не се изменят (immutable)
- ✓ Обектите от сложни (съставни) типове данни се наричат контейнери и всеки един от тях се представя като таблица от адреси, указващи къде са представени в ОП съответните техни елементи
- ✓ Някои от контейнерите се изменят (mutable), други не се изменят (immutable)

Представяне на обектите



Променливи (имена)

- ✓ Всяка променлива има име и съдържа указател (адрес) на обект от ОП
- ✓ Променливите са като контейнер с един слот
- ✓ Когато присвояваме обект на променлива, все едно променяме указателя свързан с името на променливата, да съдържа адреса на обекта в ОП
- ✓ Променливите могат да променят многократно указателя и така да се свързват с различни обекти от ОП
- ✓ Смисълът на променливите е да са имена на обекти

Пример 1

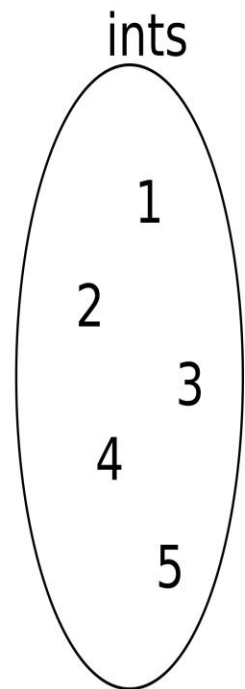
Нека имаме програмен
фрагмент:

$a = 1$

$a = a + 1$

$b = a$

$a = a + 1$



Пример 1

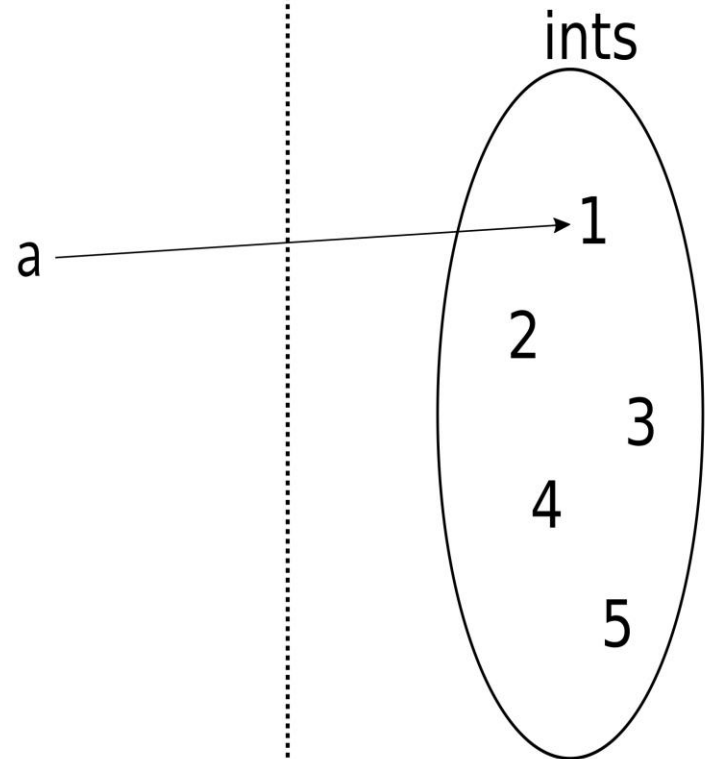
Променливата *a* се свързва с обекта 1:

$a = 1$

$a = a + 1$

$b = a$

$a = a + 1$



Пример 1

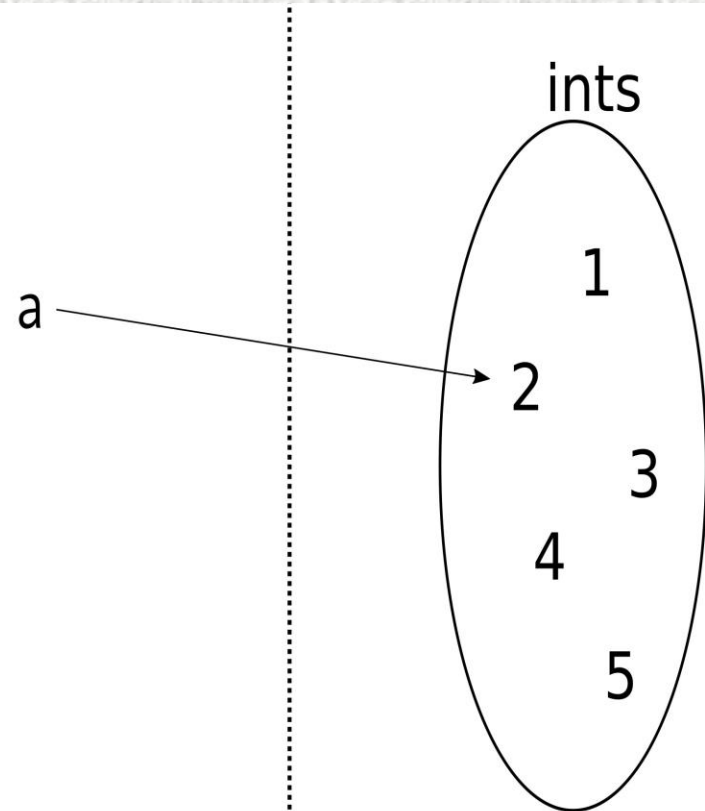
Променливата *a* се
свързва с друг обект:

$a = 1$

$a = a + 1$

$b = a$

$a = a + 1$



Пример 1

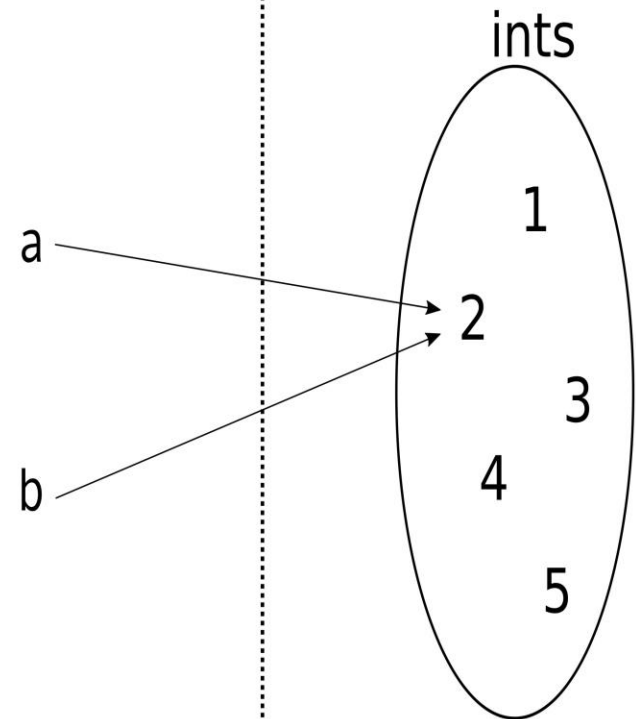
Променливата **b** се
свързва с обекта,
свързан с **a**:

$a = 1$

$a = a + 1$

$b = a$

$a = a + 1$



Пример 1

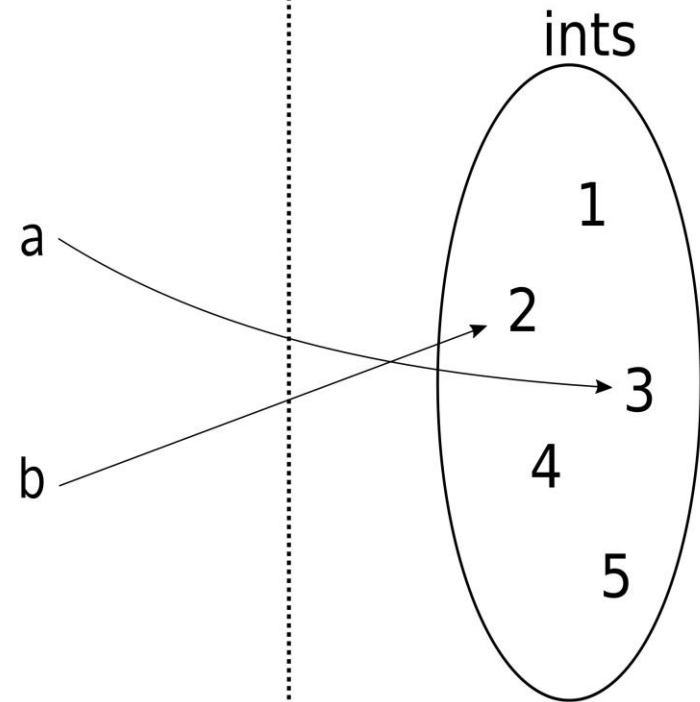
Променливата a се
свързва с нов обект

$a = 1$

$a = a + 1$

$b = a$

$a = a + 1$



Пример 2

Нека имаме програмен
фрагмент:

$a = [1, 2]$

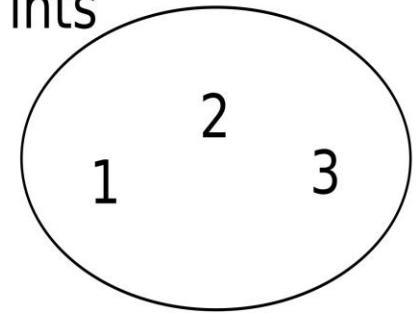
$b = [1, 1]$

$b[1] = 2$

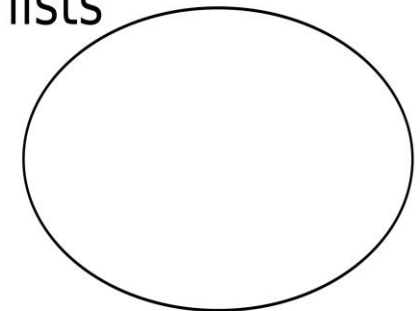
$a = b$

$a[0] = 3$

ints



lists



Пример 2

Създава се нов обект списък с елементи числата 1 и 2 и променливата `a` се свързва (указва) него

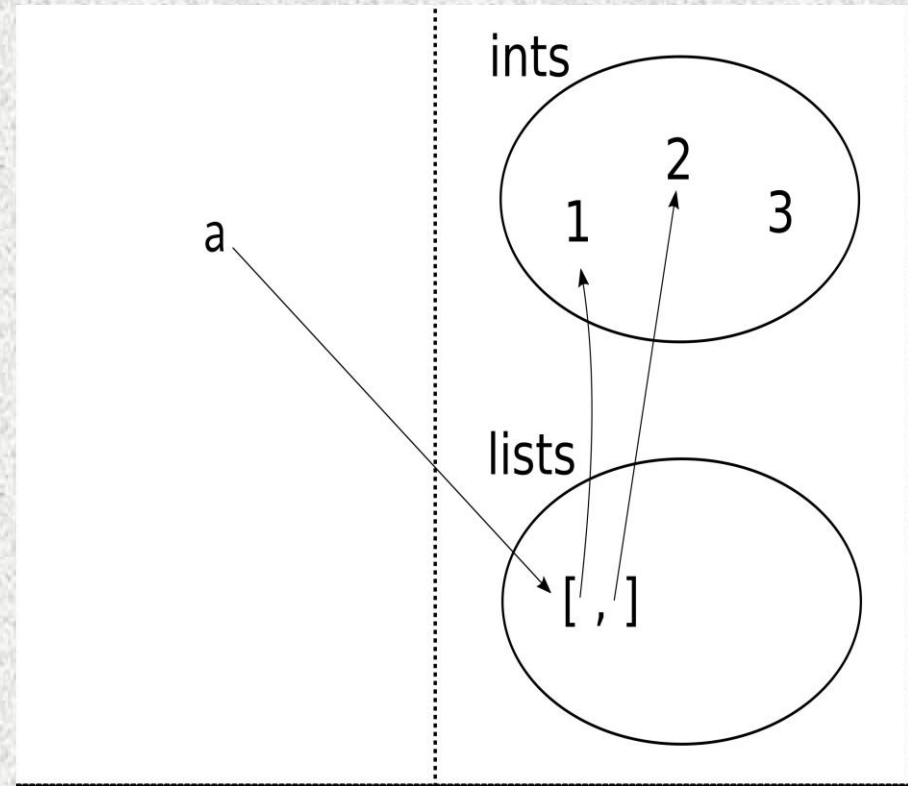
```
a = [1, 2]
```

```
b = [1, 1]
```

```
b[1] = 2
```

```
a = b
```

```
a[0] = 3
```



Пример 2

Създава се нов обект списък с елементи числата 1 и 1 и променливата **b** се свързва (указва) него

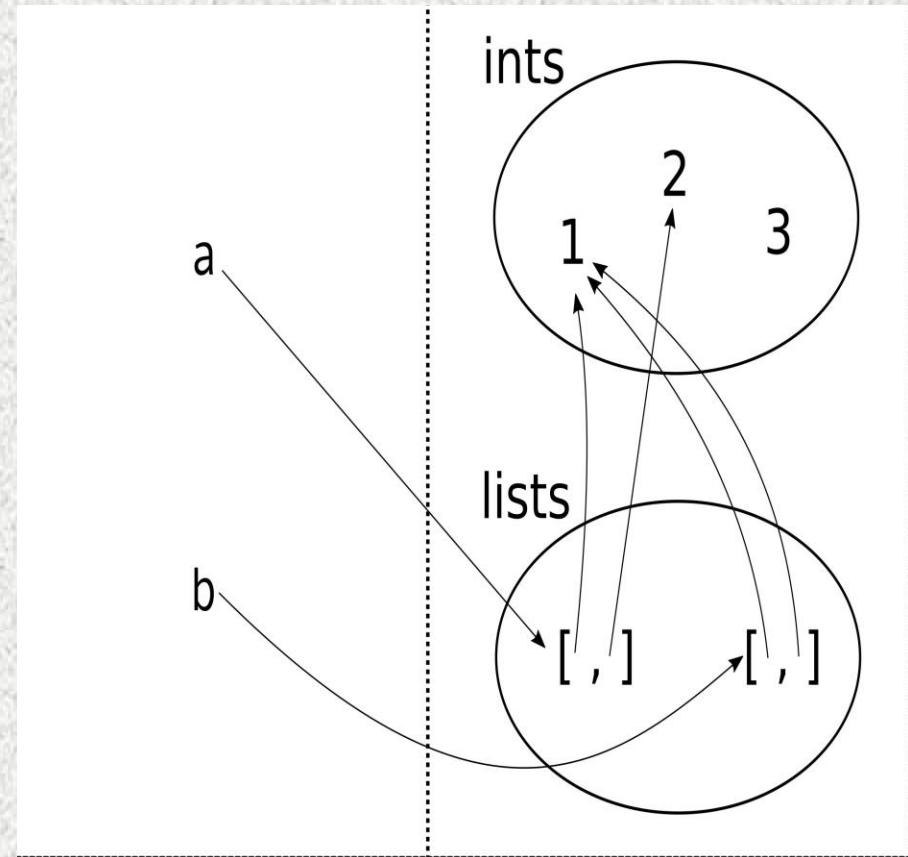
$a = [1, 2]$

$b = [1, 1]$

$b[1] = 2$

$a = b$

$a[0] = 3$



Пример 2

Променя се вторият елемент
от списъка указван от
променливата **b**

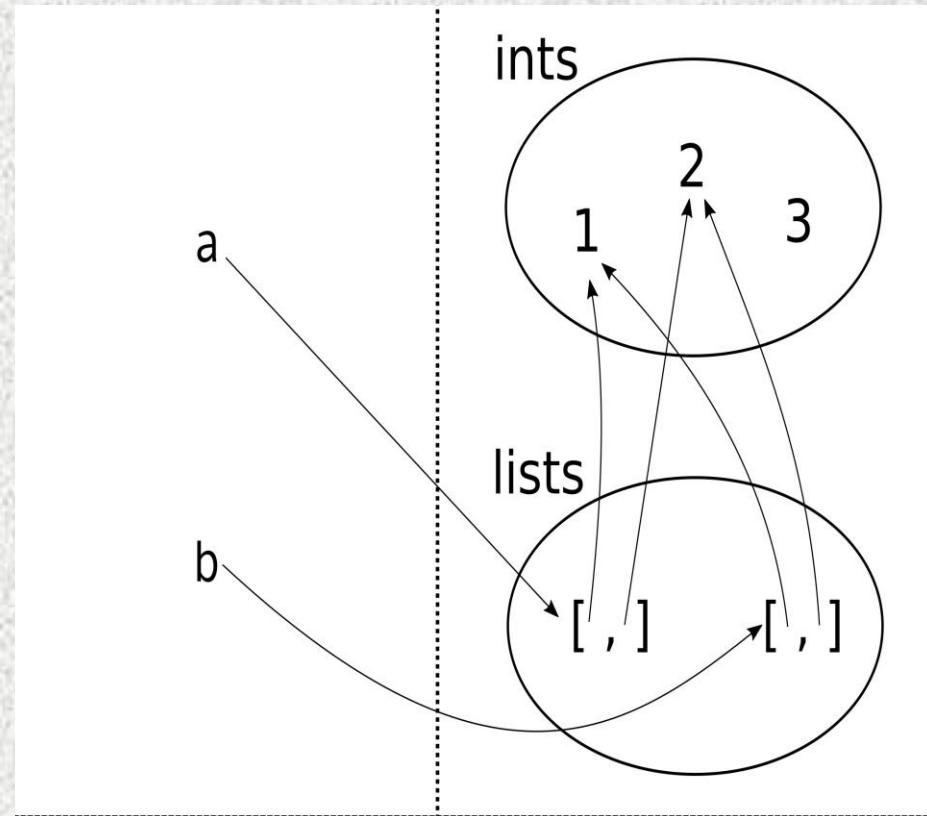
$a = [1, 2]$

$b = [1, 1]$

$b[1] = 2$

$a = b$

$a[0] = 3$



Пример 2

Променливата **a** указва
списъка указван от
променливата **b**

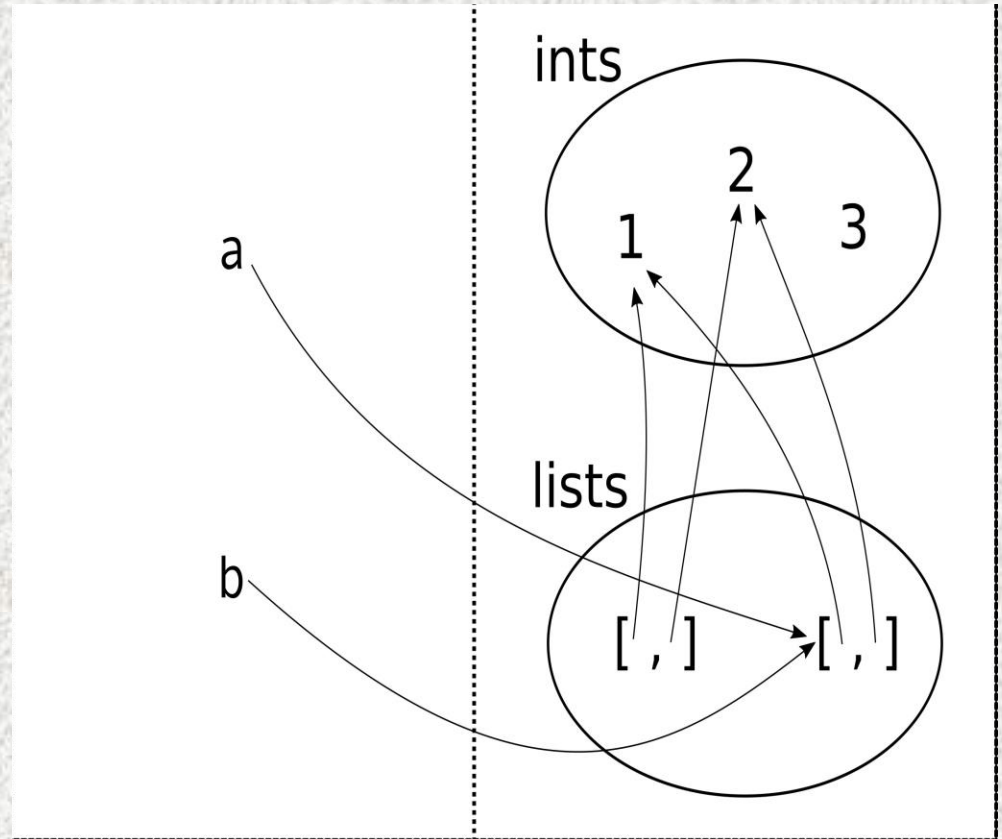
$a = [1, 2]$

$b = [1, 1]$

$b[1] = 2$

$a = b$

$a[0] = 3$



Пример 2

Списъкът указван преди
от променливата `a` вече
не е свързан с нищо и се
изтрива

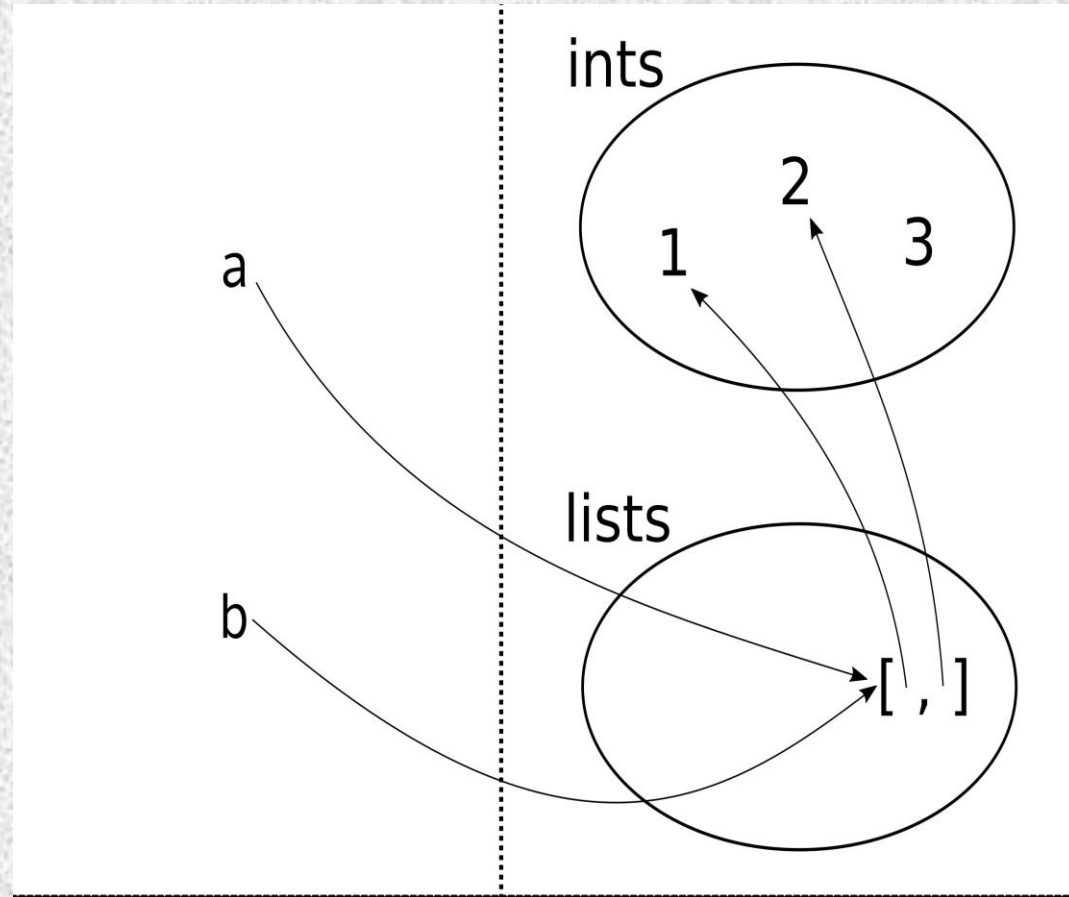
`a = [1, 2]`

`b = [1, 1]`

`b[1] = 2`

`a = b`

`a[0] = 3`



Пример 2

Списъкът указван от *a*
се променя, което води
до промяна и на *b*

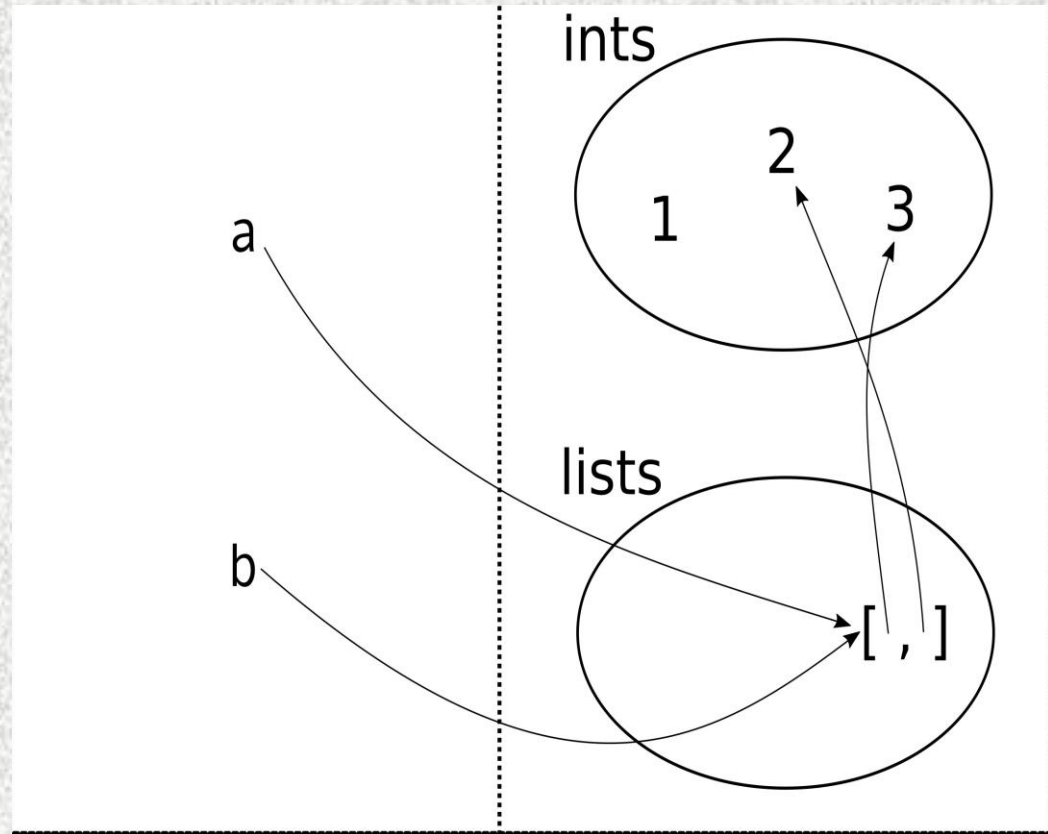
a = [1, 2]

b = [1, 1]

b[1] = 2

a = *b*

a[0] = 3



Числа

- ✓ Цели числа (по подразбиране за числа) - integer

$z = 5 // 2$ # Резултат 2, целочислено деление

- ✓ Реални числа (Floats)

$x = 3.456$

- ✓ Комплексни числа

$x = 3 + 4j$

Цели числа

Големите цели числа нямат ограничения в размера – ограничени са само от ОП

```
>>> long = 12345678901234567890123456789
>>> long ** 5
2867971861733704037813816270841549639248697656
4513250475184790028886798337811616713594453748
2406293836574832094958624542673638528386720482
949
```

Оператори за цели и реални числа

- $i + j$ → **сума**
 - $i - j$ → **разлика**
 - $i * j$ → **умножение**
 - i / j → **деление**
- При две цели резултатът е цяло
- Ако едно или двете е реално, резултатът е реално
- Резултатът е реално
-
- $i \% j$ → **остатък** при деление на i с j
 - $i // j$ → **цяла част** при деление на i с j
 - $i ** j$ → i на **степен** j

Аритметични сравнения

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

```
True
```

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

```
True
```

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

```
False
```

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

```
True
```

```
>>> 3 == 4
```

```
False
```

```
>>> 3 != 4
```

```
True
```

3 различно ли е от 4

Свързани сравнения

```
>>> 3<5<7
```

```
True
```

```
>>> 4>2>6
```

```
False
```

```
>>> 1<4>2<5>3
```

```
True
```

Просто правило: при свързани сравнения все едно, че се правят всички съседни сравнения по отделно, и на получените логически стойности се прилага функцията AND (логическо И)

Цели числа

- ✓ Представят се като неограничена последователност от нули и единици
- ✓ Могат да бъдат представени в десетична, двоична, осмична и шестнадесетична бройна система

3456

0b100110111

0o377

0xda3c

Лекция 3

Битови операции над цели числа

Операция	Резултат
$x \mid y$	Логическо ИЛИ на x и y
$x \wedge y$	Логическо XOR на x и y
$x \& y$	Логическо И на x и y
$x \ll n$	x изместен в ляво с n бита
$x \gg n$	x изместен в дясно с n бита
$\sim x$	Смяна на битовете на x (от 0 в 1 и от 1 в 0)

Битови операции над цели числа

Логическо И	# 1 ако и двата операнда са 1 # иначе е 0
Логическо ИЛИ	# 0 ако и двата операнда са 0 # иначе е 1
Логическо XOR (изключващо ИЛИ)	# 1 ако и двата операнда са различни # иначе е 0
Логическо НЕ (отрицание)	# 1 ако аргумента е 0 # 0 ако аргумента е 1

Битови операции над цели числа

>>> x = 1	# 1 десетично е 0001 двоично
>>> x << 2	# Наляво с 2 бита: 0100
4	
>>> x 2	# Логическо ИЛИ: 0011
3	
>>> x & 1	# Логическо И: 0001
1	
>>> x ^ 5	# Логическо XOR: 0100
4	

Вградени функции

```
>>> sum([3, 5, 7, 1])
```

```
16
```

```
>>> min(4, 3, 7, 2)
```

```
2
```

```
>>> max(4, 3, 7, 2)
```

```
7
```

```
>>> round(22/7, 5)
```

```
3.14286
```

```
>>> round(22/7)
```

```
3
```

Реални числа

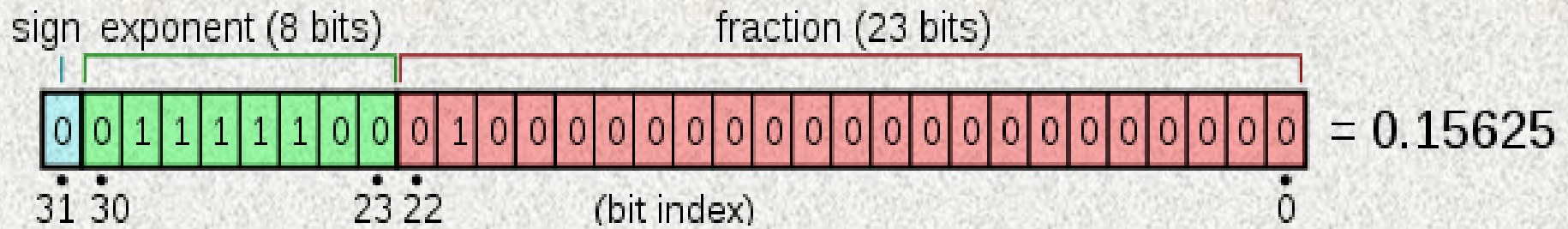
- ✓ Представят се като ограничена последователност от нули и единици (машинно зависима), задаващи цялата част и дробната част
- ✓ Изписват се най-често с помощта на десетична точка: 3.14 , 213.456
- ✓ Допустим и съкратен запис от вида: $M \text{ e } N$ (равнозначно на $M * N ** 10$)
- ✓ Например: 1e100 3.14e-10 0e0
- ✓ Вградената функция `float.as_integer_ratio()` връща 2 цели числа (a, b) -> b/a е равно на аргумента

Реални числа

- ✓ Създават се с вградената функция `float()`
- ✓ Допълнителен модул за много по-точно и приятно боравене с реални числа: `decimal`
- ✓ Допълнителен модул за работа с рационални числа: `fractions`
- ✓ Допълнителен модул за работа с тригонометрични, хиперболични и логаритмични функции: `math` (също и за работа с цели числа)
- ✓ Други вградени функции: `round()` , `abs()`, ...

Представяне на реалните числа

- ✓ Реалните числа в Python се представят в ОП като двоични числа (в модула `decimal` – като десетични)
- ✓ Всяко реално число се съхранява като $\pm M \times N$



- ✓ М се нарича мантика (significand, fraction)
- ✓ N се нарича експонента
- ✓ Използва се един бит за знак

Безкрайност

```
pos_inf = float('inf')      # плюс безкрайност
neg_inf = float('-inf')     # минус безкрайност
not_a_num = float('nan')    # NaN: Not A Number: не е число
pos_inf = math.inf
neg_inf = - math.inf
not_a_num = math.nan
math.isfinite(pos_inf)
math.isinf(pos_inf)
```


Проблеми

```
>>> 1.1 + 2.2
```

```
3.300000000000000003
```

```
>>> 0.1 + 0.1 + 0.1 - 0.3
```

```
5.551115123125783e-17
```

```
>>> 0.7 + 0.1 != 0.8
```

```
True
```

```
>>> a=1e400*0 # 1e400 = inf, inf*0 = nan
```

```
>>> a == a
```

```
False
```

```
>>>
```

Проблеми

```
>>> a = 1e30
```

```
>>> b = 1e-30
```

```
>>> c = a - a + b
```

```
>>> d = a + b - a
```

```
>>> c
```

```
1e-30
```

```
>>> d
```

```
0.0
```

```
>>>
```

Да се внимава в следните случаи

- ✓ Пресмятане на разликата между две числа, които са много близки по стойност
- ✓ Пресмятане на сума на две числа които много се различават по стойност (малкото се губи в сумата)
- ✓ Изрази които генерират плюс или минус безкрайност (`inf`) или водят до получаване на неопределена стойност (`nan`)

Преобразуване на тип данни на обект

1. Явно преобразуване (чрез вградени функции за тип данни *int*, *float*, *str*, *bool*)
2. Неявно преобразуване
 - Автоматично в изрази по време на изпълнение само между числови типове данни
 - Събиране на реално и цяло число води до получаване на цяло число
 - Преобразува се от по-прост към по-сложен тип данни (от цяло число към реално, но не обратно)
 - В логически изрази се преобразува всеки тип данни до двоичен

Примери за неявно преобразуване

```
>>> print(5 + 3.14)
```

8.14

```
>>> print(9 - 5)
```

4

```
>>> print(7/3 + 8)
```

10.333333333333334

```
>>> print(7//3 + 8)
```

10

```
>>>
```


Явно преобразуване

Примери за използване на вградените функции *int*, *float*, и *str* за явно преобразуване на обект от един тип данни в еквивалентен обект от друг тип данни

```
>>> x = 10
>>>
float(x)
10.0
>>> str(x)
'10'
>>>
```

integer → float
integer → string

```
>>> y =
"20"
>>>
float(y)
20.0
>>> int(y)
20
>>>
```

string → float
string → integer

```
>>> z =
30.0
>>> int(z)
30
>>> str(z)
'30.0'
>>>
```

float → integer
float → string

Комплексни числа

- ✓ Представят се като ограничена последователност от нули и единици (машинно зависима), задаващи реалната част и имагинерната част като реални числа
- ✓ Изписват се чрез разделител j в края: $3.14+213.456j$
- ✓ Могат да се създават с вградената функция:
`complex(real, imag)`
- ✓ Комплексно спрегнато число може да се получи с функцията: `complex.conjugate()`
- ✓ Допълнителен модул: `cmath`

Логически тип данни

- ✓ Обозначава се с вградената дума `bool`
- ✓ Съдържа две константи, които се обозначават с `True` и `False`
- ✓ Реализиран е като подмножество на типа данни `int` (цели числа)
- ✓ `True` е реализиран като `1`, а `False` е реализиран като `0`

Логически тип данни 2

```
>>> type(True)
```

```
<class 'bool'>
```

```
>>> isinstance(True, int)
```

```
True
```

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

```
True
```

```
>>> True is 1
```

```
False
```

```
>>> True + 4
```

```
5
```


Логически тип данни 3

- ✓ Всеки обект в езика Python има свойство – логически тип (True или False)
- ✓ Всяко число което е със стойност 0 има свойството False, всички други числа – True
- ✓ Всеки обект – контейнер, е False ако е празен и True в противен случай
- ✓ Обектът None има свойство False
- ✓ Обектът Notdefined има свойство True
- ✓ Вградената функция `bool(<Object>)` връща стойността на това свойство за обекта

Логически тип данни 4

- ✓ 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
```

Логически оператори

- ✓ and -> връща втория аргумент ако първият е истина, иначе връща първия аргумент
- ✓ or -> връща първия аргумент ако той е истина, иначе връща втория аргумент
- ✓ not -> има един аргумент и връща истина, ако аргумента е лъжа, или лъжа ако аргумента е истина

Логически оператори - пример

```
>>> 5 and 7
```

```
7
```

```
>>> 5 or 7
```

```
5
```

```
>>> not 5
```

```
False
```

```
>>> not 0
```

```
True
```

```
>>> '123' and '456'
```

```
'456'
```

Въведение в низовете

Определение: низът (string) е наредено множество от символи

- Може да съдържа произволни символи
- Константите от тип данни низ се ограждат с единични, двойни или тройни кавички
- ```
>>> s = "Hi there"
>>> s
'Hi there'
>>> s = "Embedded 'quote' "
>>> s
"Embedded 'quote' "
```



# Низ – литерал на няколко реда

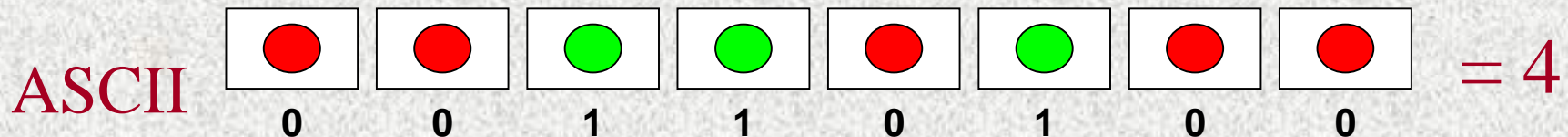
Тройни кавички при низове на повече от един ред

```
>>> s = """ a long
... string with "quotes" or
... anything else"""
>>> s
' a long\n ... string with "quotes"
or\nanything else'
>> len(s)
49
```

# Представяне на символи: код на символ

Кодовете на символите се използват за преобразуване на цифровите данни в символи, разбираеми от хората

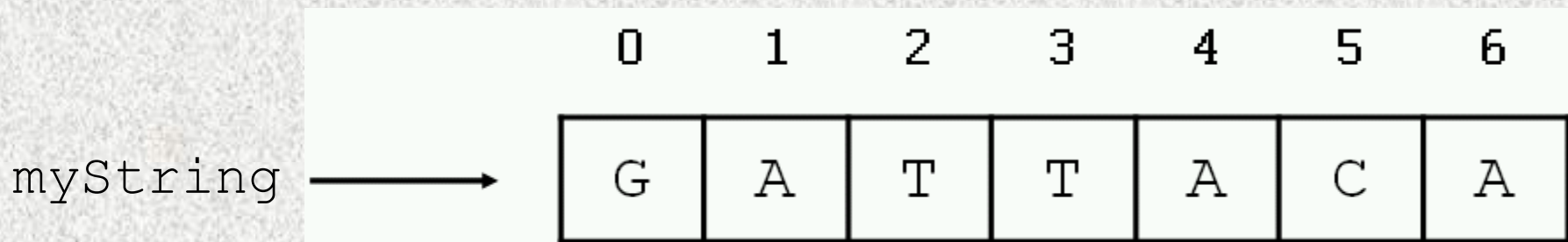
- **Американски стандартен код за обмен на информация (ASCII)** – Осем бита за един символ. Този тип кодиране се е използвал в началото от персоналните компютри
- **Extended Binary Coded Decimal Interchange Code (EBCDIC)** – Осем бита представят един символ; използвал се е в големите електронно изчислителни машини
- **Unicode** – 8, 16 или 32 бита представят един символ; над 1 милион комбинации; използва се за кодиране на повечето азбуки по света; де факто стандарт в момента. Версиите които използват променлива дължина: UTF 8, UTF 16 и UTF 32 (1, 2 или 4 байта за символ).



# Съхраняване в ОП

Всеки низ се съхранява като масив от клетки с еднакъв размер (един символ в клетка).

```
>>> myString = "GATTACA"
```



# Достъп до елементи символи

Чрез индекси в квадратни скоби

```
>>> myString = "GATTACA"
```

```
>>> myString[0]
```

```
'G'
```

```
>>> myString[1]
```

```
'A'
```

```
>>> myString[-1]
```

```
'A'
```

```
>>> myString[-2]
```

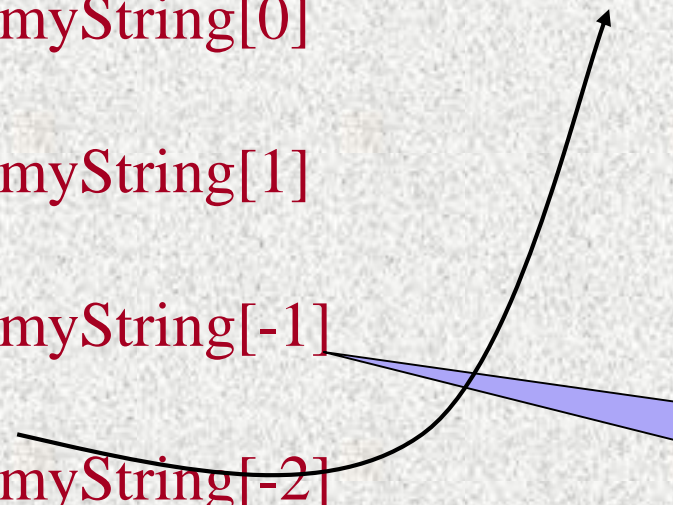
```
'C'
```

```
>>> myString[7]
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in ?
```

```
IndexError: string index out of range
```



Отрицателите индекси започват от края в дясно и се местят в ляво.

# Достъп до под-низове

```
>>> myString = "GATTACA"
```

```
>>> myString[1:3]
```

```
'AT'
```

```
>>> myString[:3]
```

```
'GAT'
```

```
>>> myString[4:]
```

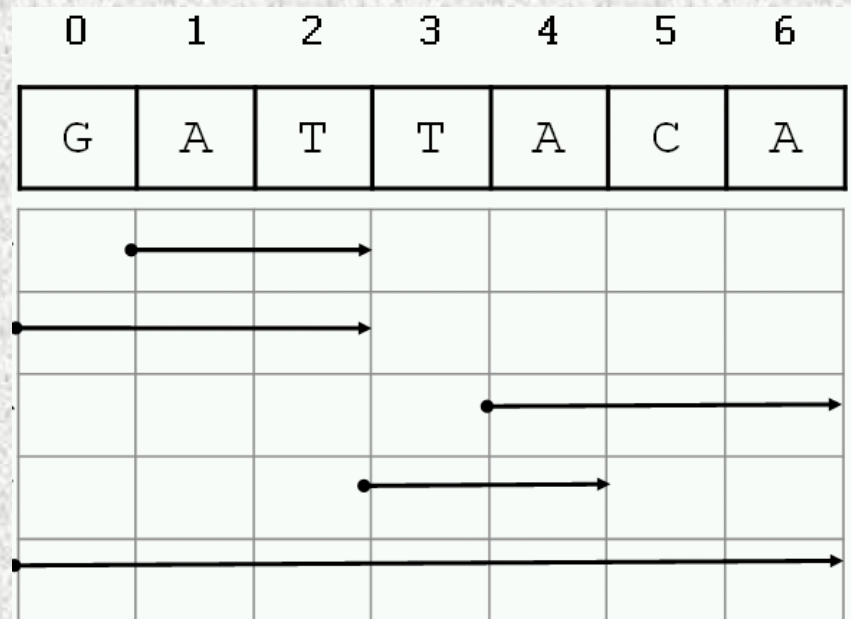
```
'ACA'
```

```
>>> myString[3:5]
```

```
'TA'
```

```
>>> myString[:]
```

```
'GATTACA'
```





# Какви символи съдържат низовете

- ✓ В низовете се съдържат символи представени най-често като Unicode (по 1, 2 или 4 байта на символ)
- ✓ Низовете могат да съдържат и байтове (произволни комбинации включващи изпълним машинен код на програми, мултимедийно цифрово съдържание и т.н.)
- ✓ В Python няма отделен тип данни за символ – един символ се представя като низ с дължина 1
- ✓ Всеки символ в даден низ има точна позиция и не може да бъде променян – низовете са `immutable` тип данни
- ✓ Низ с 0 символа се нарича още празен низ

# Видове низове и специални символи

- ✓ Escape последователност – за представяне на символ чрез неговия код (за символи за които няма клавиши на клавиатурата): `\n` (нов ред) ; `\t` (табулация) ; `\xNN` – за символ 1 байт представен с 2 16-ни числа; `\uNNNN` и `\UNNNNNNNNN` – за представяне на символи представени с 2 или 4 байта (с по 4 или 8 16-ни числа)
- ✓ Езикът Python поддържа около 100 различни схеми за кодиране на символите с 1, 2 или 4 байта, вкл. ASCII
- ✓ Всеки символ се съхранява в паметта във вътрешен формат независим от никаква кодова схема
- ✓ При въвеждане на текст от файл се отчита кодирането във файла и се прекодира до вътрешния формат

# Пример за Escape последователности

Въвеждат се със символа ‘\’ (backslash).

```
>>> "Тя каза: "Уау!""
```

```
File "<stdin>", line 1
"He said, "Wow!""
 ^
```

SyntaxError: invalid syntax

```
>>> "Тя каза: 'Уау!'"
```

```
" Тя каза: 'Уау!'"
```

```
>>> "Тя каза: \"Уау!\""
```

```
'Тя каза: "Уау!'"
```

| Escape последов. | Смисъл         |
|------------------|----------------|
| \\               | Backslash      |
| \'               | Кавичка        |
| \"               | Двойни кавички |
| \n               | Нов ред        |
| \t               | Табулация      |



## Други видове низове

- ✓ „суров“ (raw) низ – игнорира Escape последователност – например стандартен низ “C:\new\_Python\_dir\text.py” няма да позволи достъп до файла - \n ще се възприеме като нов ред, а \t като табулация. Затова:  

```
>>> p=r“C:\new_Python_dir\text.py”
‘C:\\new_Python_dir\\text.py’
```
- ✓ Последен символ \ - комбинацията \” вкарва кавичката в низа и той не е завършен! Решава се с дублиране на \
- ✓ Байт низ –съдържа байтове несвързани със символи и кодови таблици (за машинен код или несимволна информация): слага се пред тях буква ‘b’ – b’sp\xc4m’
- ✓ Unicode низ –започва с ‘u’ - u’sp\xc4m’

# Оператори за низове

```
>>> animals = "Cats " + "Dogs "
```

```
>>> animals += "Rabbits"
```

```
>>> print(animals)
```

```
Cats Dogs Rabbits
```

```
>>> salute = 'Hi! ' * 3
```

```
>>> print(salute)
```

```
Hi! Hi! Hi!
```

```
>>> print('-' * 80)
```

```

```

```
>>> 'ack' in 'hacker'
```

```
True
```

```
>>> 'ace' in 'hacker'
```

```
False
```

```
>>> del salute
```

```
len('Човек')
```

```
>>> 5
```



# Оператори за низове

## ✓ Конкатениране (слепване)

- `word = 'HeIp' + x`
- `word = 'HeIp' 'a'`
- `word = 'HeIp' * 4`

## ✓ Поднизове

- `'Hello'[2] → 'l'`
- Парче (slice): `'Hello'[1:3] → 'el'`
- `word[-1] → последен символ`
- `len(word) → дължина на низ – брой символи`
- `immutable`: не може да се променя стойност на елемент в низ.

# Още за парчета от низове

```
>>> word = 'Help'+'a'
```

```
>>> word
```

```
'Helpa'
```

```
>>> word[1:]
```

```
'elpa'
```

```
>>> word[:-1]
```

```
'Help'
```

```
>>> word[:]
```

```
'Helpa'
```

```
>>> word[1:4:2]
```

```
'ep'
```

```
>>> word[::2]
```

```
'Hla'
```

# Функции за преобразуване на низове

```
>>> int('123')
```

```
123
```

```
>>> str(213)
```

```
'213'
```

```
>>> repr(690)
```

```
'690'
```

```
>>> ord('ю')
```

```
1102
```

```
>>> chr(1089)
```

```
'с'
```

```
>>> S='123'
```

```
>>> S=S+'123'
```

```
>>> S
```

```
'123123'
```

# Използване на низове

- ✓ Обикновено се съхраняват във файлове
- ✓ Всеки файл с текст има собствено кодиране на символите от низовете (ASCII, Unicode, ...)
- ✓ За обработка на текстовете, символите от файловете се зареждат в ОП – Python превежда всеки символ от начина на кодиране във файла в неговия вътрешен формат на кодиране
- ✓ След края на обработката символите се записват от ОП във външни файлове, като от междинния формат се кодират в избран за файла метод на кодиране

# Обхождане на низове

Чрез цикъл while:

```
n = len(str)
```

```
i = 0
```

```
while i < n:
```

```
 print(str[i], end=' ')
```

```
 i += 1
```

Чрез цикъл for:

```
for i in str:
```

```
 print(i, end=' ')
```



## Обхождане на низове 2

Чрез оператор за slicing (ляво на дясно):

```
for i in str[::]:
 print(i, end="")
```

Чрез оператор за slicing (дясно на ляво):

```
for i in str[::-1]:
 print(i, end="")
```

# Проверка дали низ се съдържа в друг

Чрез оператор **in**

```
str = input("Въведи низ: ")
sub = input(" Въведи втори низ : ")
if sub in str:
 print(sub+" е открит в първия низ")
else:
 print(sub+" не е открит в първия низ ")
```

# Проверка дали низ се съдържа в друг

Чрез метод `find`

```
str = input("Въведи низ: ")
sub = input(" Въведи втори низ : ")
n = str.find(sub, 0, len(str))
if n == -1:
 print(" вторият низ не е открит в първия низ")
else:
 print(" вторият низ е открит @: ", n + 1)
```

# Проверка дали низ се съдържа в друг

Чрез метод `count`

```
str = "New Delhi"
```

```
n = str.count("Delhi")
```

```
str = "New Delhi"
```

```
n = str.count("e", 0, 3)
```

```
str = "New Delhi"
```

```
n = str.count('e', 0, len(str))
```

# Форматиране на низове

- ✓ Вграден метод за форматиране:  
`<низ> = <форматиране>.format(<аргументи>)`
- ✓ Елементът `<форматиране>` задава начина на форматиране на аргументите
- ✓ Елементът `<аргументи>` задава реалните обекти, които ще бъдат форматирани
- ✓ `<низ>` ще съдържа посочените от `<аргументи>` обекти, форматирани в низ както е указано в елемента `<форматиране>`
- ✓ Като правило този вграден метод (функция) се използва за извеждане на данни на екран или запазване във файл



# Примери за форматиране

```
id, name, sal = 10, "Иван", 120
```

```
print("{ }, { }, { }".format(id, name, sal))
```

```
print("{ }-{}-{}".format(id, name, sal))
```

```
print("ID: {0}\tName: {1}\tSal: {2}\n".format(id, name, sal))
```

```
print("ID: {2}\tName: {0}\tSal: {1}\n".format(id, name, sal))
```

```
print("ID: {two}\tName: {zero}\tSal: {one}\n".format(zero=id,
 one=name, two=sal))
```

```
print("ID: {:d}\tName: {:s}\tSal: {:.10.2f}\n".format(id, name, sal))
```

# Примери за форматиране

```
>>> "{0} / {1} / {2}".format(27, 2, 2018)
```

```
'27 / 2 / 2018'
```

```
>>> "day/month/year".format(day=27, month="02", year=2018)
```

```
'day / month / year'
```

```
>>> "{day}/{month}/{year}".format(day=27, month="02", year=2018)
```

```
'27 / 02 / 2018'
```

```
>>> "{0:<12}' '{1:^12}' '{2:>12}'".format('Ден', 'Месец', 'Година')
```

```
'''Ден ' ' Месец ' ' Година'''
```

```
'''{0:*<12}' '{1:->12}'".format('Ден','Година')
```

```
'''Ден*****' '-----Година'''
```

# Форматиране на низове с литерали

- ✓ Указва се с поставяне на 'f' или 'F' преди низа:  
f<низ> или F<низ>
- ✓ Обектът <низ> съдържа текстов литерал, в който са вмъкнати в { } изрази, които се оценяват. Изразите могат да съдържат променливи, както и метод за кодиране и начин на форматиране
- ✓ След кодирането и форматирането полученият низ се извежда (или присвоява)
- ✓ Кодиране се задава с '!s' (извършва се със str()), '!r' (извършва се с repr()), и '!a' (чрез ascii()).
- ✓ Форматиране се указва с ':'

# Примери за форматиране

```
>>> name = "Иван"
```

```
>>> f"Той се казва {name!r}."
```

```
"Той се казва 'Иван'."
```

```
>>> width = 10
```

```
>>> precision = 4
```

```
>>> value = decimal.Decimal("12.34567")
```

```
>>> f"резултат: {value:{width}.{precision}} " # nested fields
```

```
'резултат: 12.35'
```

```
>>> today = datetime(year=2020, month=2, day=4)
```

```
>>> f"{today:%B %d, %Y}" # using date format specifier
```

```
'February 4, 2020'
```

```
>>> number = 1024
```

```
>>> f"{number:#0x}" # using integer format specifier
```

```
'0x400'
```



# Преобразуване

| Формат<br>СИМВОЛ | Преобразуване до                   |
|------------------|------------------------------------|
| %c               | СИМВОЛ                             |
| %s               | Низ чрез str                       |
| %i               | Цяло число със знак                |
| %d               | Цяло число                         |
| %u               | Цяло число без знак                |
| %o               | Осмично число                      |
| %x               | 16-чно число с малки букви         |
| %X               | 16-но число с ГОЛЕМИ букви         |
| %e               | Реално с експонента с малки букви  |
| %E               | Реално с експонента с ГОЛЕМИ букви |
| %f               | Реално с плаваща точка             |
| %g               | комбинация от %f и %e              |



## Лекция 3

# Специални символи

| Символ | Функционалност                                                          |
|--------|-------------------------------------------------------------------------|
|        |                                                                         |
| *      | Аргумент за брой знаци или точност                                      |
| -      | Ляво подравняване                                                       |
| +      | Дясно подравняване                                                      |
| <sp>   | Интервал преди положително число                                        |
| #      | Добавяне на водещи 8-ни или 16-ни нули                                  |
| 0      | Изравняване в ляво с 0 вместо с интервали                               |
| %      | '%%' вместо '%'                                                         |
| Var    | Съпоставя с аргументите на променливата                                 |
| m.n.   | m е минимален общ брой знаци, n е общ брой цифри след десетичната точка |

# Три типа низове

- ✓ `str` – използва се за представяните в паметта Unicode символи
- ✓ `bytes` – използва се за представяне в паметта на двоични данни (машинен код, графични изображения и други)
- ✓ `bytearray` – също като `bytes` за представяне на двоични данни в паметта, но `mutable`
- ✓ Трите типа низове имат общо множество от операции, но с различно действие

# Методи за работа с низове

- ✓ В Python, метод е функция, която е дефинирана за даден обект (или клас от обекти).
- ✓ Синтаксис на обръщение към метод:  
`<object>.<method>(<parameters>)`

```
>>> dna = "ACGT"
```

```
>>> dna.find("T")
```

```
3
```

# Примери за методи

```
>>> "GATTACA".find("ATT")
1
>>> "GATTACA".count("T")
2
>>> "GATTACA".lower()
'gattaca'
>>> "gattaca".upper()
'GATTACA'
>>> "GATTACA".replace("G", "U")
'UATTACA'
>>> "GATTACA".replace("C", "U")
'GATTAUA'
>>> "GATTACA".replace("AT", "**")
'G**TACA'
>>> "GATTACA".startswith("G")
True
>>> "GATTACA".startswith("g")
False
```

# Заключение

- ✓ Идентичност на обекти
- ✓ Цели, реални и комплексни числа
- ✓ Вградени функции за работа с числа
- ✓ Какво представляват низовете
- ✓ Как се представят и кодират символите във файловете
- ✓ Какво е Escape последователност
- ✓ Видове низове
- ✓ Вградени оператори и команди
- ✓ Вградени методи и функции