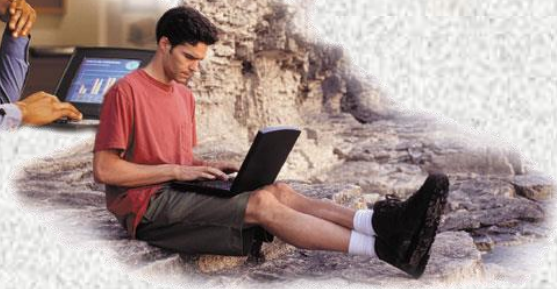




Основи на Програмирането



Лекция 2

Въведение в езика Python

Какво ще научите

- ✓ Какво представляват основните типове данни
- ✓ Какви са основните команди
 - За присвояване
 - За повторение
 - За проверка
 - За вход и изход
- ✓ Как се осъществяват основните операции над базовите типове от данни
- ✓ Как се присвояват стойности на променливи
- ✓ Как се преобразуват данни от един тип в друг

Програми на PYTHON

- **програма** е последователност от дефиниции и команди
 - Дефинициите са в началото на програмите, включват специфичен набор от команди, които се **оценяват (компилират)**, и могат да се използват по-нататък в програмата като команди (пример функция)
 - Командите се **изпълняват** от интерпретатора на Python в команден шел (прозорец)
- **командите** указват на интерпретатора специфични действия, които да се извършат на момента
- Програмите могат да се задават директно в командния прозорец и да се изпълняват команда по команда, или се подават като **файл**, който се изчита от интерпретатора и изпълнява ред по ред (команда по команда)

Структура на програма на Python

- ✓ Всяка програма започва с дефиниции, основно на функции, класове и други обекти на Python или на други езици за програмиране, най-често C, които да се вмъкнат в текущата програма
- ✓ Всяка програма съдържа редове с команди
 - На всеки ред по правило има една команда
 - Някои команди се разполагат на повече редове
 - Командите се изпълняват последователно (по редове)
 - Всяка команда съдържа изрази, включващи данни (обекти) и оператори, и различни ключови думи, които указват семантиката на командата

Изрази и обекти

- ✓ Всяка команда съдържа изрази
- ✓ Всеки израз включва данни (обекти) и оператори, указващи действията над данните ($3 + 5$)
- ✓ Основни елементи: променливи и константи
- ✓ Променливата именува обект от данни от някакъв тип (число, символ, множество и т.н.)
- ✓ Константата задава името и типа на обекта — например 7 (числото 7), “7” (символът 7) и т.н.
- ✓ Всеки израз се оценява до някаква стойност, която също представлява обект от някакъв тип данни
- ✓ Тази стойност се използва в командата в зависимост от нейната семантика (нейният смисъл)

Програмен фрагмент

```
x = 17 + 23          # коментар
y = "Здравей"       # друг коментар
z = 3.14
if z == 3.14 or y == "Здравей":
    x = x + 1
    y = y + " свят"  # свързване
print(x)
print(y)
```

Кратки бележки

- Изместването влияе върху смисъла на програмата:
 - С еднакво изместване се указва програмен блок
- При първо присвояване се създава променлива
 - Променливите не се обявяват (декларират)
 - Python определя типа данни динамично
- За присвояване **=** ; за сравнение **==**.
- Оператори за числа: **+** **-** ***** **/** **//** **%**
 - Операторът **+** се използва и за обединяване на низове
- Логическите оператори са думи (**and**, **or**, **not**), не символи
- Извеждане на информация: с функцията **print()**.

Променливи

Не се декларира (описват) - направо се използват, най-често в команда за присвояване:

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

Проверяваме за стойност на променлива, като зададем името и. Това предизвиква извеждане на стойността (защото променливата е израз, който се оценява)

```
>>> a
```

```
1
```

Променливите са указатели към данни в паметта – промяна на стойност на променлива означава ново свързване (променливата се свързва с друг обект)

Променливи (2)

- Променливите трябва да имат стойност (да са свързани с обект) преди да се използват, иначе:

```
>>> b
```

```
Traceback (innermost last):
```

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

```
NameError: name 'b' is not defined
```

```
>>>
```

- При настъпване на всякакви грешки, Python използва механизъм на обработка на грешките

Променливи (3)

Всеки обект, с който е свързана променлива, винаги има тип:

```
>>> a = 1
>>> type(a)
<class 'int'>
>>> a = "Hello"
>>> type(a)
<class 'string'>
>>> type(1.0)
<class 'float'>
```

Присвояване

- ✓ Присвояване се задава с оператора =
- ✓ В лявата част обикновено има име на променлива, а в дясната част израз
- ✓ Изразът се оценява и стойността се присвоява на променливата
- ✓ Команди за присвояване:

```
>>> size = 40
```

```
>>> a = b = c = 3
```

Проверка за равенство

- Проверка за равенство между два обекта в израз се прави с `==`
- При проверка за равенство може да се направи преобразуване на типа на обектите от двете страни на знака за сравнение

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

```
True
```

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

```
True
```

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

```
False
```


Прости типове от данни

- Числа (цели числа, реални числа, комплексни числа)
- Символни низове
- Логически константи

Числа

- ✓ Цели числа (по подразбиране за числа) – тип `int`

`z = 5 // 2` # Отговор 2, целочислено делене

- ✓ Реални числа – тип `float`

`x = 3.456`

- ✓ Комплексни числа – тип `complex`

`x = 3 + 4j`

Цели числа

Целите числа нямат ограничения в размера

Ограничени са само от ОП

```
>>> long = 12345678901234567890123456789
>>> long ** 5
28679718617337040378138162708415496392486976
56451325047518479002888679833781161671359445
37482406293836574832094958624542673638528386
720482949
```

Реални числа

```
>>> 3.1472
```

```
>>> 2 + 1.0
```

```
>>> 1e12
```

```
>>> 100000000 > 1e7
```

```
True
```

```
>>> 1000000 < 1e7
```

```
True
```

```
>>> 10000000 == 1e7
```

```
True
```


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()`

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`

Арифметични оператори

```
>>> 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
```

Оператори за цели и реални числа

- $i + j$ → **сума**
 - При две цели резултата е цяло
- $i - j$ → **разлика**
 - Ако едно или двете е реално, резултата е реално
- $i * j$ → **умножение**
 - Ако едно или двете е реално, резултата е реално
- i / j → **деление**
 - Резултата 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

Арифметични сравнения

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

```
True
```

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

```
False
```

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

```
True
```

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

```
True
```

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

```
False
```

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

```
True
```

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)

Логически оператори

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

```
>>> b = False
```

```
>>> a and b
```

```
False
```

```
>>> a or b
```

```
True
```

```
>>> not b
```

```
True
```

```
>>> a and not (b or c)
```

```
False
```

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

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. $=$, $+=$, $-=$, $*=$, $/=$, $//=$, $\%=$, $**=$

Низове

- Един низ може да съдържа произволни символи
- Всяка константа низ се задава с използване на единични, двойни или тройни кавички като разделител за начало и край

```
>>> s = "Hi there"
>>> s
'Hi there'
>>> s = "Embedded 'quote' "
>>> s
"Embedded 'quote' "
```

Низове на повече от един ред

Тройни кавички се използват за задаване на низове на повече от един ред

```
>>> s = """ a long
... string with "quotes" or
anything else"""
>>> s
' a long\n ... string with "quotes"
\nor anything else'
>>> len(s)
```

Оператори за низове

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

```
>>> animals += "Rabbits" # добавяне със слепване
```

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

Cats Dogs Rabbits

```
>>> fruit = ', '.join(['Apple', 'Banana', 'Orange'])
```

```
>>> print(fruit)
```

Apple, Banana, Orange # прилагане на вградена функция

```
>>> date = '%s %d %d' % ('Feb', 20, 2018)
```

```
>>> print(date)    # форматиране и извеждане
```

Feb 20 2018

```
>>> name = '%(first)s %(last)s' % {'first': 'Apple', 'last': 'Microsoft'}
```

```
>>> print(name)
```

Apple Microsoft

Достъп до елементи от низове

✓ Конкатениране (слепване)

➤ `x = 'a'`

➤ `word = 'HeIp' + x # 'HeIpa'`

✓ Поднизове

➤ `word[2] → 'l'`

➤ Парче (slice): `word[1:3] → 'el'`

➤ `word[-1] → последен символ -> 'a'`

➤ `len(word) → дължина на низ`

➤ `immutable`: не може да се променя стойност на елемент в низ.

Вградени типове данни

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

Идентичност на обекти

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

```
True
```

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

```
False
```

```
>>> 1 and True
```

```
True
```

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

```
False
```

```
>>> bool(1) == True
```

```
True
```

```
>>> bool(False)
```

```
False
```

```
>>> bool(True)
```

```
True
```


Изменяеми (Mutable) / Неизменяеми (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 си

Списъци

- ✓ Контейнери съдържащи елементи (обекти)
- ✓ 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']`
- ✓ 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`

Примери за програмиране

```
>>> a, b = 0, 1  
>>> while b < 10:  
    print(b)  
    a, b = b, a + b
```

1

1

2

3

5

8

**Управление на
последователността на
изпълнение на 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 for

```
>>> for x in range(10):    #0-9  
        print(x)
```

Извежда числата 0 ... 9 по 1 на ред

```
>>> fruits = ["Apple", "Orange"]  
>>> for fruit in fruits:  
        print(fruit)
```

Apple

Orange

Цикъл While

```
>>> x = 0
>>> while x < 100:
    print(x)
    x += 1
```

Извежда числата до 100 по 1 на ред

Примерна програма (в IDLE)

```
x = 34 - 23          # коментар
y = "Hello"         # Още един
z = 3.45
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World" # Слива низове
print(x)
print(y)
```

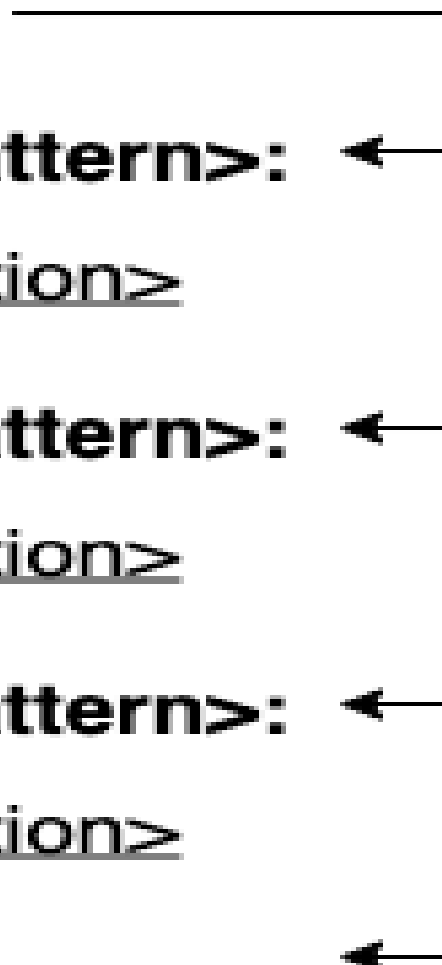

Усложнени проверки с if

```
x = int(input("Please enter #:"))
if x < 0:
    x = 0
    print('Negative changed to zero')
elif x == 0:
    print('Zero')
elif x == 1:
    print('Single')
else:
    print('More')
```

- ✓ има команда MATCH (за първи път във версията на езика 3.10)

Структурни проверки с match

match subject:



```
case <pattern>: <action>  
case <pattern>: <action>  
case <pattern>: <action>  
case _: <action wildcard>
```

Структурни проверки с match

```
# status – съдържа код на завършване  
# на изпълнението на http заявка  
match status:  
    case 400:  
        return "Лоша заявка"  
    case 404:  
        return "Не открит ресурс"  
    case 418:  
        return "Програмно зависим"  
    case _:  
        return "Проблем в Интернет"
```

Структурни проверки с match

point е редица от 2 елемента (x, y)

match point:

case (0, 0):

print("Начало")

case (0, y):

print(f"Y={y}")

case (x, 0):

print(f"X={x}")

case (x, y):

print(f"X={x}, Y={y}")

case _:

raise ValueError("Не е точка")

Припомняне

- ✓ Използване на интервали за оформяне на текста
 - Интервалите определят блоковете команди
- ✓ Първото присвояване на променлива я създава
 - Типа на променливата не се декларира
 - Python определя типа на обектите от логиката
- ✓ Присвояването е `=` и сравнението е `==`
- ✓ Операторите `+` `-` `*` `/` `%` се очакват за изрази с числа
 - Операторът `+` се използва за сливане на низове и `%` за тяхното форматиране (както в командата `printf` в C)
- ✓ Логически оператори се задават с думи (`and`, `or`, `not`) а не със символи
- ✓ Основна команда за извеждане на данни е `print`

Интервал

Интервалът и новия ред имат специално значение в Python:

✓ Нов ред се използва за разграничение между команди

Ползва се \ за преход към нов ред

✓ Блоковете команди не се разделят с { } а с подходящо използване на интервали

- Първият ред с *по-малко* интервали е извън блока от команди

- Първият ред с *повече* интервали започва нов блок

✓ Двоеточие започва нов блок в много команди
(дефиниция на функция, клауза then)

Коментари

- ✓ Коментари започват с #, останалата част от реда се игнорира
- ✓ Те са добър стил за документиране на програми; средства като debugger я използват често

```
def fact(n):  
    """fact(n) подразбира че n е  
    положително цяло число и връща като  
    резултат числото n! """  
    return 1 if n==1 else n*fact(n-1)
```

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 създава обект в първия момент в който се появява в лявата част на команда за присвояване:
`x = 3`
- ✓ Всеки обект и връзка се изтриват след края на използването им

Правила за имена

- ✓ Имената отчитат големи/малки букви и не могат да започват с цифра. Съдържат букви, цифри, и символ за подчертаване.

bob Bob _bob _2_bob_ bob_2 BoB

- ✓ Запазени думи:

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

Съглашения за имената

Python общността използва следните съглашения за имената на обектите в езика:

- ✓ Малки букви със „_“ за имена на функции, методи и атрибути
- ✓ Малки букви със „_“ или само големи букви за константи
- ✓ Думи без „_“ започващи с големи букви при имена на класове и методи
- ✓ Атрибути: интерфейс, _вътрешни, __частни

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
```

Достъп до несъществуващ обект

Достъп до име на обект, който още не е създаден, води до грешка при изпълнение

```
>>> y
```

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

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

```
NameError: name 'y' is not defined
```

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

```
>>> y
```

```
3
```

Обекти - константи

- `int` – представя **цели числа**, например 5
- `float` – представя **реални числа**, например 3.27
- `bool` – представя **логически** стойности `True/False`
- Вградена команда `type()` извежда типа на обект:

```
>>> type(5)
```

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

```
>>> type(3.14)
```

```
<class 'float'>
```

Преобразуване на типове данни

- позволява **преобразуване на обект of един тип в друг**
- `float(3)` преобразува цяло число 3 в реално 3.0
- `int(3.9)` преобразува реалното число 3.9 в цялото 3
- `bool(7)` преобразува цялото число 7 в булева константа `True`

Извеждане на данни

Функция `print` за показване стойност на данни и изрази

```
>>> print(3+2)
```

5

```
>>> print(3+2.0)
```

5.0

```
>>> print('This is sentence"s output.')
```

This is sentence"s output.

```
>>> print("This is sentence's output.")
```

This is sentence's output.

```
>>> print(7 == True)
```

False

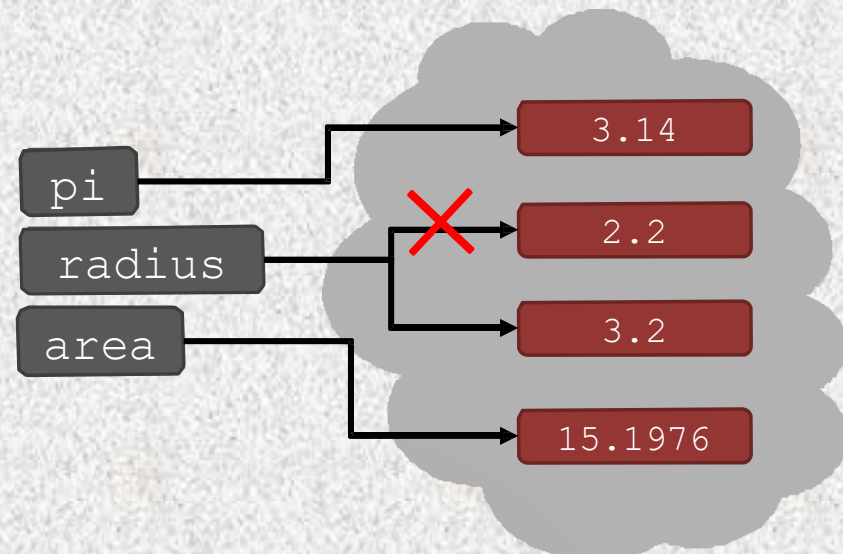
```
>>> print(1 == True)
```

True

Промяна стойността на променливи

- Чрез нова команда за присвояване
- Предишната стойност все още е в паметта, но няма връзка към нея
- Стойността на променливата area не се променя докато не се изпълни нова команда за присвояване

```
pi = 3.14  
radius = 2.2  
area = pi*(radius**2)  
radius = radius+1
```



Заключение

- Програмите на Python включват различни команди и обекти (типове данни)
- Променливите получават стойност с команди за присвояване
- Изразите включват променливи, константи и оператори, и се оценяват до някаква стойност
- Данните имат различен тип
- Данни от един тип могат да се преобразуват до данни от друг тип
- Условните оператори се използват за управление на реда на изпълнение на командите
- Командите за повторение позволяват блок от команди да се изпълни многократно