Introduction to Python

About me

Trayan Iliev

– CEO of **IPT – Intellectual Products & Technologies**
http://www.iproduct.org

– Oracle® certified programmer 15+ Y

– end-to-end reactive fullstack apps with Java, ES6+, TypeScript, Angular, React and Vue.js

– 12+ years IT trainer: Spring, Java EE, Node.js, Express, GraphQL, SOA, REST, DDD & Reactive Microservices

– Voxxed Days, jPrime, Java2Days, jProfessionals, BGOUG, BGJUG, DEV.BG speaker

– Organizer RoboLearn hackathons and IoT enthusiast

What Will You Learn?

- Типове данни и операции – 5 h
- Оператори и изключения – 5 h
- Функции – 8 h
- Модули и пакети – 12 h
- Работа с файлове – 8 h
- Паралелизиране – 12 h
- Обектно ориентирано програмиране – 20 h
- Тестване – 10 h

Course Schedule

- Block 1: 09:00 – 11:00
- Pause: 10:40 – 11:00
- Block 2: 11:00 – 13:20

Where to Find The Code and Materials?

https://github.com/iproduct/intro-python

Python Example

```
"""factorial done recursively and iteratively"""
def fact1(n):
    ans = 1
    for i in range(2, n + 1):
        ans = ans * i
    return ans

def fact2(n):
    if n < 1:
        return 1
    else:
        return n * fact2(n - 1)

if __name__ == '__main__':
    print(fact1(100))
    print(fact2(100))
```

Why Python

- Easy to learn scripting language – flat learning curve
- Very popular in cloud computing, data science and machine learning communities
- You can make really short programs for complex things
- The final code of your node is quite easy to read and understand.
- You can do anything with Python. Python is a very powerful language with libraries for anything you want.
- It is easier to integrate it with web services based on Flask or Django.
- Running even on constrained Arduino devices - e.g. MicroPython on the ESP32 IoT platform

Short History of Python

- Създаден в Холандия, началото на 90-те, от Guido van Rossum (Гуидо ван Росум)
- Името идва от Monty Python
- Създаден като проект с отворен код
- Създаден като скриптов език, но постепенно се превръща в нещо повече
- Проектиран като лесно разширяем, обектно - ориентиран и функционален език
- Използван от Google в началото
- Все по-популярен и известен

Short History of Python

- “Python е експеримент за това колко свобода е нужна на един програмист.
- Твърде много свобода – никой няма да може да разбира програмите на другите.
- Твърде малко свобода – няма да има достатъчно ясни и изразителни програми.”

  - Guido van Rossum

How Popular is Python

- От края на 2014 г. езикът Python е най-използван като първи език за обучение по програмиране в топ университетите на САЩ:
- в 8 от топ 10 департаменти преподаващи компютърни науки (80%)
- в 27 от топ 39 (69%).
- Избран от IEEE за най-популярен език за програмиране за 2017 г.
- Най-популярен за 2018 г. според Гугъл

Python Philosophy - I

The Zen of Python, by Tim Peters

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.

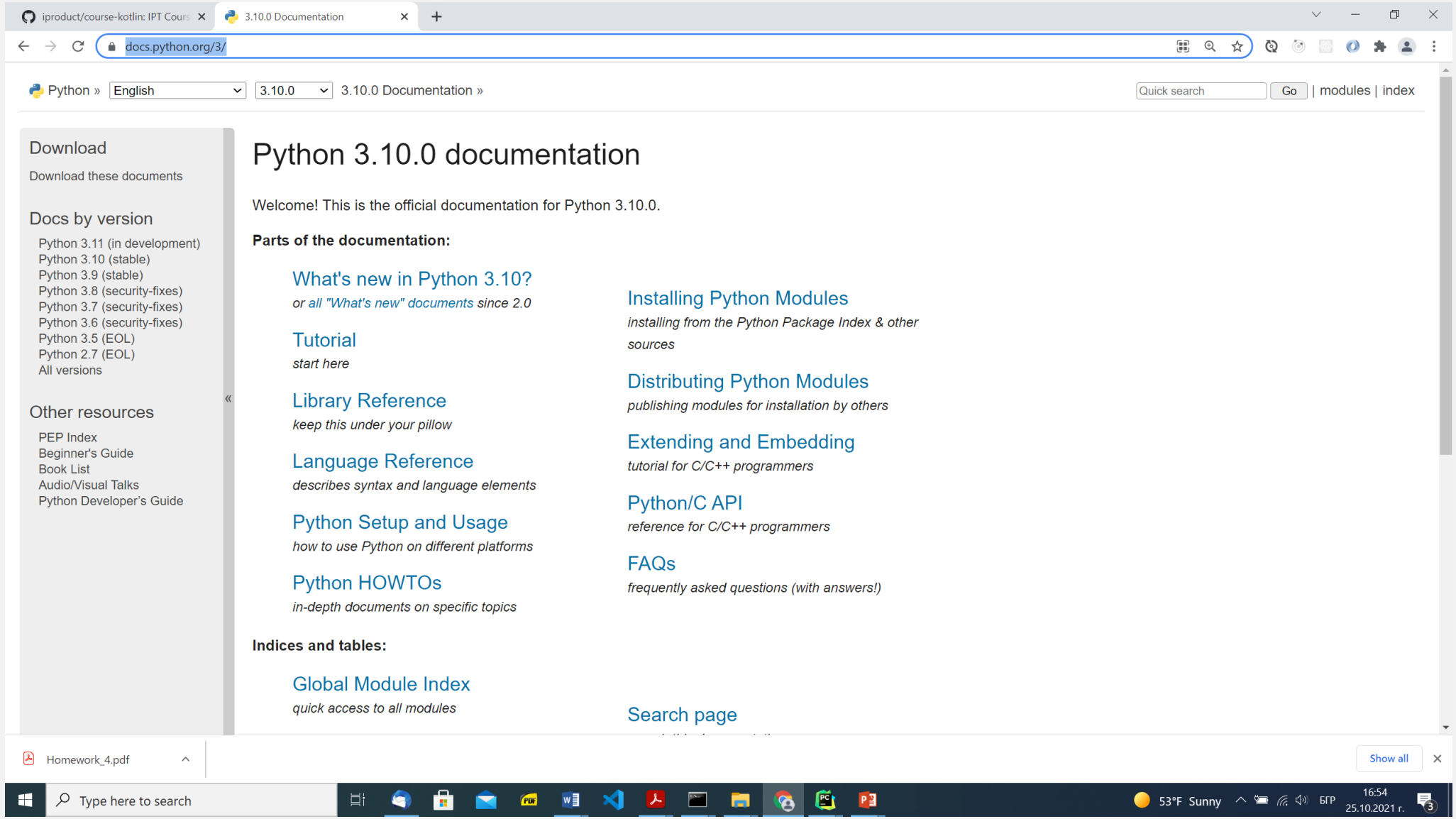
Python Philosophy - II

- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

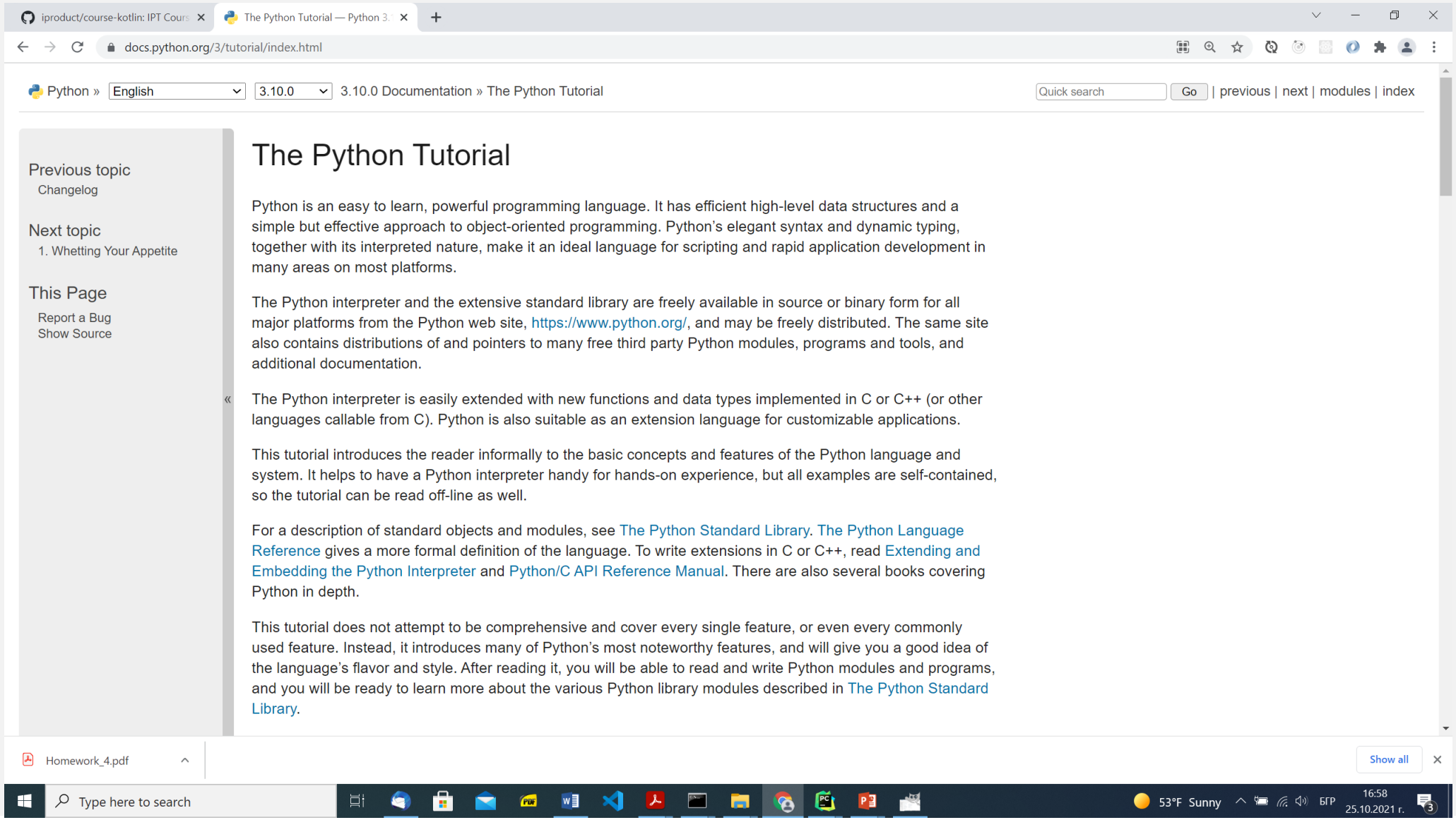
Python vs. Java

- Текстът на програмата е 5-10 пъти по-кратък
- Динамични типове от данни
- Много по-бързо създаване на програма
- Няма нужда от предварителна компилация
- По-кратък текст
- Изпълнява се по-бавно
  - Компенсира се с вградени модули
- Може да се ползват заедно Python с Java: Jython!

http://docs.python.org/



https://docs.python.org/3/tutorial/index.html



Python Interpreter

- Предлагат се интерпретатор и компилатор
- Интерактивен интерфейс:

```
[finin@linux2 ~]$ python
Python 2.4.3 (#1, Jan 14 2008, 18:32:40)
[GCC 4.1.2 20070626 (Red Hat 4.1.2-14)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> def square(x):
...   return x * x
...
>>> list(map(square, range(1, 5)))
[1, 4, 9, 16]
```

Python Scripts

- Когато се изпълни програма на Python във файл, командния ред на интерпретатора оценява всяка команда и дефиниция последователно
- ОС предоставя допълнителен механизъм за задаване на аргументи към програмата (скрипта) чрез командния ред, както и за пренасочване на входните и изходните данни от програмата
- Езикът Python има механизъм който указва на една програма дали и как да функционира като скрипт или модул, който да се използва от друга програма

Expressions and Objects

- Всяка команда съдържа изрази
- Всеки израз включва данни (обекти) и оператори, указващи какви действия да се извършат над данните (3 + 5)
- Основен елемент на изразите: променливи и константи
- Променливата се използва за именуване на обект от данни от някакъв тип (число, символ, множество и т.н.)
- Константата задава директно името и типа на обекта – например 7 (числото 7), “7” (символът 7), {7} (множество съдържащо един елемент – числото 7)
- 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)

Variables

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

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

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

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

Variables - II

- Objects always have a type:

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

Assignment

- 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

Equality Check

- 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
```

Simple Data Types

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

Strings

- 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'"
```

Multiline Strings

- 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" or anything else'
>>> len(s) 45
```

Integer Numbers

- Целите числа нямат ограничения в размера
- Ограничени са само от ОП

```
>>> long = 123456789012345678901234567890
>>> long ** 5
28679718617337040378138162708415496392486976564513250475184790028
88679833781161671359445374824062938365748320949586245426736385283
86720482949
```

Arithmetic Operators

```
>>> 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
```

Arithmetic Comparisons

```
>>> 5 > 3
True
>>> 3 >= 5
False
>>> 3 < 5
True
>>> 3 <= 5
True
>>> 3 == 5
False
>>> 3 != 5
True
```

Logical Operators

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

Objects Identity

```
>>> 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
```

String Operators

```
>>> 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
```

Accessing Individual Characters

Конкатениране (слепване)

word = 'Help' + x

word = 'Help' 'a'

Поднизове

'Hello'[2] → 'l'

Парче (slice): 'Hello'[1:3] → 'el'

word[-1] → последен символ

len(word) → дължина на низ

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

Lists

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

Lists

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

Programming Examples

```
a, b = 0, 1
>>> while b < 10:
        print(b)
        a, b = b, a + b
1
1
2
3
5
8
```

If – elif – else

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

For Loop

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


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

While Loop

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

More Complex Checks

```
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‘)
```

Basic Datatypes

Цели числа (по подразбиране за числа) - integer

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

Реални числа (Floats)

```
x = 3.456
```

Низове (Strings)

Използват “” или ‘’ - например “abc” == ‘abc’

Един разделител може да се използва между другите: “matt’s”

Използвайте тройни кавички за низове на няколко реда или ако съдържат едновременно единични и двойни кавички в тях

Space

Интервалът и новия ред имат специално значение в Python:

Нов ред са 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)

Comments

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 n! """
    return 1 if n==1 else n*fact(n-1)
```

Names

Имената отчитат големи/малки букви и не могат да започнат с число. Съдържат букви, числа, 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

Name Conventions

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

Особена форма на използване на големи букви при имена на класове и методи

Атрибути: интерфейс, _вътрешни, __частни

Assignment

За няколко променливи

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

Размяна на стойности:

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

Верижно присвояване

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

Assignment

За няколко променливи

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

Размяна на стойности:

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

Верижно присвояване

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

Containers

- 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 на контейнер е от тип mutable, стойността му може да се промени.

String Containers

- Всеки контейнер от тип низ включва в себе си произволен брой елементи от тип символ

Пример: 'hello'

- Стойността на един низ включва стойностите на отделните му елементи символи
- Контейнерът от тип низ е immutable, а това означава, че нито един негов елемент може да се промени. Всеки елемент от един низ е символ, който също е immutable, стойността му не може да се промени.

Embedded Types

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')

Lists

- Елементите се задават в квадратни скоби, отделени с ','
- 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

Lists 2

- Могат да бъдат с произволна и променлива дължина, която може да се разбере с вградената функция **len()**
- Всеки елемент на списъка може да бъде обект от произволен тип данни, включително контейнер, и в частност списък
- Всеки обект от тип списък (**list**) е mutable - може да се променя (за разлика от низовете)
- Списъците наподобяват масивите в математиката но за разлика от тях допускат като елементи обекти от произволен тип (масивите съдържат обекти от един тип данни)

List Operators

- Допустими са същите както и за низове, 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]
>>> ['Ni!'] * 4
['Ni!', 'Ni!', 'Ni!', 'Ni!']
```

List Indexing

- Начинът на индексиране и достъп до елементи или под-списъци е същият както при низовете
- Всеки елемент на списък може да се променя (за разлика от низовете)

L[first:last:step] # всяко first, last, step може да го няма

- По премълчаване: first=0, last=-1, step=1

L[0] е първи елемент, а L[-1] е последният елемент

L[:] -> L[0:-1] # еквивалентно на L

L[::-1] -> еквивалентно на обърнат списък

Multi-Dimensional Lists

- Достъпът до много мерни списъци е както при многомерните масиви в математиката:
- L[i][j] – в двумерния списък L обозначава (j+1)-я елемент в списък, който е (i+1)-и елемент в L
- Аналогично се обобщава за произволна размерност

L = [[[1,2,3], 4, 5, 6], [7, 8, 9]]

L[1][2] -> 9 ; L[0][1] -> 4 ; L[0][0][1] -> 2

L[i][j][k] – задава k+1-и елемент в списък, който е вложен j+1-и елемент в друг списък, който е вложен i+1-и елемент в списъка L

Predefined List Functions

- Допустими са същите както и за низове, имат на практика същото действие (len, in, del, enumerate …)
- Функцията list се използва за преобразуване на обекти от произволен тип в списъци

```
>>> print(list('spam'))
['s', 'p', 'a', 'm']
>>> print(list(range(-4,4)))
[-4, -3, -2, -1, 0, 1, 2, 3]
```

- Функцията range(first, last, step) връща обект итератор съдържащ числата first .. last през step

Predefined List Methods

L.append(4)	# добавя елемента 4 в края на L
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

Examples 1

```
>>> L = [1, 2, 3]
>>> L.append(4)
>>> L
[1, 2, 3, 4]
>>> L.extend([5, 6, 7])
>>> L
[1, 2, 3, 4, 5, 6, 7]
>>> L.insert(3,3)
>>> L
[1, 2, 3, 3, 4, 5, 6, 7]
>>> L.index(5)
5
>>> L.count(3)
2
```

Predefined List Methods 2

```
L.sort(reverse=True|False, key=myFunc)  # сортира списъла L по
                                        # резултата връщан от myFunc
L.reverse()                             # обръщане
L.copy()                                # копиране
L.clear()                               # изтрива всички елементи
L.pop(i)                                # изтрива i+1-я елемент и го
                                        # връща като стойност (по подразбиране: -1)
L.remove(X)                             # изтрива първото срещане на X в L
```

Examples 2

```
>>> M=L.sort()
>>> L, M
([1, 2, 3, 3, 4, 5, 6, 7], None)
>>> L.reverse()
>>> L
[7, 6, 5, 4, 3, 3, 2, 1]
>>> M = L.copy()
>>> M
[7, 6, 5, 4, 3, 2, 1, 0]
>>> M = L.copy()
>>> M
[7, 6, 5, 4, 3, 2, 1, 0]
```

Examples 3

```
>>> M
[7, 6, 5, 4, 3, 2, 1, 0]
>>> print(M.pop(3))
4
>>> M
[7, 6, 5, 3, 2, 1, 0]
>>> M.pop()
0
>>> M.remove(5)
>>> M
[7, 6, 3, 2, 1
```

Examples 4

```
>>> L = [1, 2, 3, 4, 5]
>>> id(L)
47768800
>>> id(L[1:4])
47767760
>>> L[1:4]
[2, 3, 4]
>>> L[1:4:-1] = [4, 3, 2]
>>> L
[1, 4, 3, 2, 5]
```

Dictionaries

- Речниците са неподредено 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 (immutable) може да има само една стойност
- Различните ключове трябва да са с различна стойност

Dictionaries Usage Examples

```
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
```

Dictionaries Usage Examples 2

- D = {'име': 'Bob', 'age': 40} # речник с 2 елемента
- D['име'] = 'Bobby' # смяна на стойност
- D['pos'] = ‘boss‘ # добавя нов елемент
- del(D ['age‘]) # изтрива елемент
- >>> D = dict(name='Bob', age=40)
- >>> D
- {'name': 'Bob', 'age': 40}
- >>> E = dict.fromkeys(['име', 'възраст'])
- >>> E
- {'име': None, 'възраст': None}

Dictionaries – Predefined Functions

```
>>> zip(['key1', 'key2'], [123, "Name"])
<zip object at 0x0017CCB0>
>>> dict(_)
{'key1': 123, 'key2': 'Name'}
>>> len(_)
2
>>> E = zip(['key1', 'key2'], [123, "Name"])
>>> del(E['key1'])
>>> print(E)
{'key2': 'Name'}
```

Dictionaries – Predefined Methods

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.popitem()	# връща и изтрива случайна 2-ка

Dictionaries – Predefined Methods 2

```
D.get(key)                          # връща обекта за ключа key
                                    # ако няма такъв, връща none
D.get(key, <default>)               # връща обекта за ключа key
                                    # ако няма такъв, връща <default>
D.pop(key)                          # връща обекта за ключа key и изтрива
                                    # двойката от речника (ако няма – none)
D.pop(key, <default>)               # като горе, ако няма: <default>
D.setdefault(key)                   # аналогична на get
D.setdefault(key,<default>)         # ако няма ключ key,
                                    # вмъква двойка key:<default>
```

Dictionary Comprehensions

```
>>> D = {x: x*2 for x in range(10)}
>>> D
{0:0, 1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}

>>> D = {c: c * 3 for c in 'STRING'}
>>> D
{'S': 'SSS', 'T': 'TTT', 'R': 'RRR', 'I': 'III', 'N': 'NNN', 'G': 'GGG'}
```

Dictionary Examples

```
>>> D = {'a': 1, 'b': 2, 'c': 3}
>>> K = D.keys()
>>> V = D.values()
>>> I = D.items()
>>> K
dict_keys(['a', 'b', 'c'])
>>> list(K)
['a', 'b', 'c']
>>> list(V)
[1, 2, 3]
```

Dictionary Examples 2

```
>>> D = {'a': 1, 'b': 2, 'c': 3}
>>> del D['b']
>>> list(K)
['a', 'c']
>>> list(V)
[1, 3]
>>> list(I)
[('a', 1), ('c', 3)]
```

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 индексиране 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)

Tuples 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)

Tuple Operators

- 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', ', ')
```

Creating Tuples

- Празна редица се създава като: ()

```
>>> empty = ()
>>> type(empty)
<class 'tuple'>
```

- Редица от един елемент се задава като се постави след него запетайка: t = 3,
- Присвояването t=(<обект>) присвоява на t директно обекта <обект>, а не създава редица от 1 елемент!

```
>>> s = (4)
>>> type(s)
<class 'int'>
```

Creating Tuples 2

- Пакетирано създаване

```
>>> point1 = 2, 3
>>> point2 = 3, 2
>>> coord = point1, point2  # ((2, 3), (3, 2))
```

- Автоматично де-пакетиране

```
>>> x, y = coord          # x = (2,3) ; y = (3, 2)
>>> a1, a2 = x            # a1 = 2  ; a2 = 3
```

Predefined Tuple Functions

- Допустими са същите както за низове и списъци, имат същото действие (len, in, del, enumerate)

```
>>> type((1, 2, 3))
<class 'tuple'>
>>> T = tuple('String')
>>> T
('S', 't', 'r', 'i', 'n', 'g')
>>> T = tuple([1, 2, 3])
>>> T
(1, 2, 3)
```

Mutating Tuple Elements

```
>>> T=2, 4, [1, 2, 5]
>>> T
(2, 4, [1, 2, 5])
>>> T[1] = 0
Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
...
>>> T[2][1] = 0
>>> T
(2, 4, [1, 0, 5])
```

Error Handling in Python

- Когато по време на изпълнение на програма възникне грешка (опит за достъп до несъществуващ обект или за извършване на несъществуваща операция към дадени обекти), програмата се прекъсва, всичко създадено до момента в ОП се губи, и се дава съобщение за мястото и причината на грешката.
- В следващите слайдове ще разглеюдаме механизъм и средства за нормална обработка на грешките, така че програмата да не губи всичко създадено до момента, и да завърши нормално.
- Изключението (exception) е събитие, което прекратява нормалното изпълнение на програмата. Всяка грешка предизвиква изключение.
- При настъпване на изключение, изпълнението на програмата се прекратява, и управлението се предава на програма за обработка на изключения.

Errors and Exceptions

- Syntax Errors:

```
>>> while True print('Hello world')
  File "<stdin>", line 1
    while True print('Hello world')
                   ^
SyntaxError: invalid syntax
```

- Exceptions

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Exception Handling

- Стандартната програма за обработка на изключения действа на ниво главен модул и нейните съобщения се виждат когато изпълним програма с грешки.
- Всеки програмист също може да предизвиква изключения, както и да създава собствена програма за обработка на изключенията.
- Има огромен списък от грешки предизвикващи изключения.
- Програмистът може да дефинира свои собствени изключения.
- Всички изключения в Python са представени като йерархия от класове
- Някои изключения са частен случай (под-клас) на други.

Exception Examples

```
>>> print(7/0)
Traceback (most recent call last):
  File "<pyshell#122>", line 1, in <module>
    print(7/0)
ZeroDivisionError: division by zero
>>> print([1,2] - [3,4])
Traceback (most recent call last):
  File "<pyshell#123>", line 1, in <module>
    print([1,2] - [3,4])
TypeError: unsupported operand type(s) for -: 'list' and 'list'
```

Sample Exceptions

IndexError	индекс на редица (контейнер) който не е валиден
KeyError	липсващ ключ за речник
ZeroDivisionError	деление на 0
ValueError	неподходяща стойност за вградена функция
TypeError	използване на функция/операция върху грешен обект
IOError	входно/изходна грешка

За повече вградени изключения виж:
https://docs.python.org/3/library/exceptions.html

Exceptions Hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- EncodingWarning
           +-- ResourceWarning
```

try/except/else/finally

- Изключенията са събития, които могат да променят нормалната последователност на изпълнение на програмата.
- Те се генерират автоматично при грешки или други причини.
- **try/except** : вградени команди за обработка на изключения в Python
- **try/finally**: специална клауза за последна обработка след реагирането на всички потенциални грешки
- **raise**: команда за генериране на изключение
- **assert**: условно включване на изключение

Exception Processing Example

```
>>> 3 / 0
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    3 / 0
ZeroDivisionError: division by zero
>>> def div(x,y):
        try:
            return x/y
        except ZeroDivisionError:
            return "Деление на 0"
```

```
>>> div(7,3)
2.3333333333333335
>>> div(4,0)
'Деление на 0'
```

Exception Processing

```
try:
        <блок команди>              #където очакваме грешки
except <име1>:                      #обработка на изключение <име1>
        < блок команди >
except <име2> as <пром>:            #запаване на изключение като променлива
        < блок команди >
except (<name3>,<name4>):           #обработка на няколко изключения
        < блок команди >
except:                             #обработка на всички други (catch-all)
        < блок команди >
else:                               # изпълнява се ако няма грешки
        < блок команди >
finally:                            # изпълнява се винаги (closing, clean-up)
        < блок команди >
```

try/except/else/finally

- В един блок **try:** трябва да има поне една клауза **except:**
- Клауза **except** без израз (име) се допуска, и тя задължително е последна
- Клаузата **else:** не е задължителна
- Клаузата **finally:** не е задължителна, гарантирано се изпълнява
- Ако в някоя клауза се получи друга грешка, тя не се обработва с тази команда **try**, а се търси следваща нагоре в йерархията на имената команда **try**, или ако няма – системната програма за обработка на грешки

try/except/else/finally ex.

```
def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("division by zero!")
...     else:
...         print("result is", result)
...     finally:
...         print("executing finally clause")
...
```

```
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

Rasing a New Exception, Custom Exceptions

- **raise** - предизвиква явно някакво изключение
- **raise <име>** #предизвиква изключение с <име>
- **raise <име> from <име1>** #указва <име1> като причина за <име>
- **raise** # предизвиква отново последното
- Като име на изключение се задава или вграден, или дефиниран от потребителя клас (в този случай трябва да бъде дефиниран като подклас на вградения клас Exception)

```
>>> class MyError(Exception):
        def __init__(self,*args):
            Exception.__init__(self,*args)
```

Rising Exceptions & Inheritance Example

```
class B(Exception):
    pass


class C(B):
    pass


class D(C):
    pass
```

```
for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
```

try/except/else/finally ex.

```
def divide(x, y):
...     try:
...         result = x / y
...     except ZeroDivisionError:
...         print("division by zero!")
...     else:
...         print("result is", result)
...     finally:
...         print("executing finally clause")
...
```

```
>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

assert

- assert е специална условна форма на командата raise
- Синтаксис:

assert <проверка> [, <съобщение>]

- Това е еквивалентно на

if not <проверка> raise AssertionError()

или на

if not <проверка> raise AssertionError(<съобщение>)

- Смисълът е да се провери някакво условие и само ако то не е изпълнено, да се предизвика системното изключение AssertionError

assert example

```
>>> def fun(x,y):
        assert x>0, 'x трябва да да бъде положително'
        assert y<0, 'y трябва да бъде отрицателно'
        return y ** x
>>> fun(-4, 2)
16
>>> fun(-4, -2)
Traceback (most recent call last):
  File "<pyshell#130>", line 1, in <module>
    fun(4, -2)
  File "<pyshell#128>", line 3, in fun
    assert x>0, ‘x трябва да бъде положително'
AssertionError: x трябва да бъде положително
```

Exception Objects

- Всички видове изключения са дефинирани като класове
- Съществува йерархия от всички възможни изключения в езика Python
- Всяко конкретно изключение (грешка) е обект от съответния клас
- Достъп до всеки обект може да се получи чрез атрибута **as** в дефиницията на клаузата за обработката на конкретното изключение:

except <име> **as** <пром>: #обработка с променлива

< блок команди >

File "<pyshell#128>", line 3, in fun

assert x>0, 'x трябва да бъде положително'

AssertionError: x трябва да бъде положително

Receiving Exception Object Info

- **sys.exc_info()** - връща за всеки обект-изключение редица (tuple) с три елемента:
- type: от кой клас е обекта - изключение
- value: стойността на обекта - изключение
- traceback: копие от стека за изпълнение с информация за йерархията на областите довели до изключението

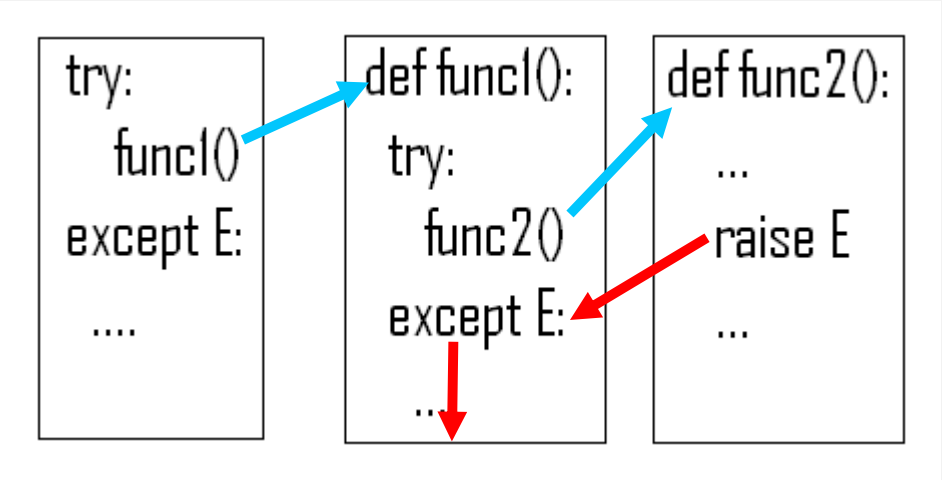
Predefined Exception Classes

- **Exception** – суперклас за всички изключения
- **StandardError** – суперклас за всички вградени изключения
- **ArithmeticError** – суперклас за всички изключения свързани с аритметични операции
- **OverflowError** – под-клас дефиниращ конкретен вид изключение

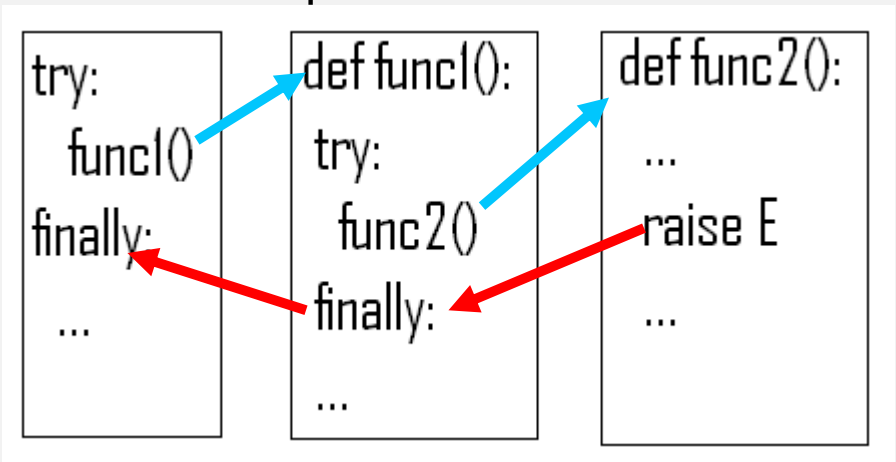
За повече вградени изключения виж:
https://docs.python.org/3/library/exceptions.html

Exception Processing Hierarchy

- Щом се открие кой да обработи грешката, тя се премахва:



- finally не премахва exception:



Programming Problem: Numerology

Напишете програма, която при въвеждане на рождена дата (напр. 12.05.1982) да изчислява личната година според нумерологията, ако знаете че:

1. Личната година=сбор от цифрите на въведения месец+сбор от цифрите на въведения ден +сбор от цифрите на текущата година
2. Полученото се редуцира докато не стане едноцифрено
3. Според получената лична година изведете подходяща прогноза като използвате например данните от този сайт: http://umaybg.com/?p=17

Problem: Books Invoicing

Даден е списък с книги, в който всяка книга е преставена като редица (идентификатор, име на книга, автори, издателство, година, цена):

```
books = [
    (1, "Learning Python", "", "Марк Лътз, Дейвид Асър", "O'Reily", 1999, 22.7),
    (2, "Think Python", "An Introduction to Software Design", "Алън Б. Дауни","O'Reily", 2002, 9.4),
    (3, "Python Cookbook", "Recipes for Mastering Python 3", "Браян К. Джоунс и Дейвид М. Баазли", "O'Reily", 2011, 135.9)
    ]
```

1. Да се напише функция, която приема списъка с книги като аргумент, форматира го и го извежда в конзолата под формата на таблица с фиксирана ширина на всяка колона, достатъчна да побере данните от съответното поле на всяка от книгите. Да се демонстрира работата на функцията. (10 точки)

2. Да се пресметне и отпечати общата сума на цените на всички книги в списъка и върху тази цена да се начсли ДДС (20%). Да се отпечатят начисления ДДС и крайната цена с ДДС на всички книги в списъка. (10 точки)

Problem: Numbers

Да се напише програма, която по въведени цели числа A, B (A< B), N и последователност от N на брой цели числа, извежда на екрана елементите на последователността в следния ред: отначало всички числа, които са по-малки от A, след това всички числа в интервала [A, B] и накрая всички останали числа, запазвайки техния първоначален ред.

Problem: Walking a Tree Represented as List

- Напишете функция на езика Python, която по подадено дърво, листата на което са низове (strings) връща низ (string) съдържащ конкатенирани (долепени) всички стрингове (листа). Дървото е представерно като списък с елемети подсписъци (клонове на дървото) и низове (листа на дървото).

- Демонстрирайте правилната работа на реализираната функция с примерни данни.

Problem: Primes by Sieve of Eratosthenes

Напишете програма, която при въвеждане на число **n** да отпечатва на конзолата всички прости числа по-малки или равни на n по метода на Ератостен.

Решето на Ератостен е алгоритъм за намиране на всички прости числа в интервала [1, n], където n е произволно естествено число. Алгоритъмът е кръстен на древногръцкия математик Ератостен, на когото е и приписано изобретяването му.

Идеята е следната: ако намерим просто число n, то всяко n-то след него няма да е просто.

За повече информация и анимирана демонстрация виж Wikipedia

Problem: Bulls and Cows

Напишете програма за игра с компютъра на Бикове и крави - логическа игра за отгатване на цифри. Играе се от двама противника, като всеки се стреми да отгатне тайното число, намислено от другия. След всеки ход, противникът дава броя на съвпаденията.

Играта протича по следния начин. На лист хартия всеки участник написва своето тайно число. Тайните числа са четирицифрени, като цифрите не трябва да се повтарят. След това, последователно един след друг, играчите задават въпрос с предположение за числото на противника. Противникът отговаря, като посочва броя на съвпаденията – ако дадена цифра от предположението се съдържа в тайното число и се намира на точното място, тя е „бик“, ако е на различно място, е „крава“.

Пример:

Тайно число: 4271

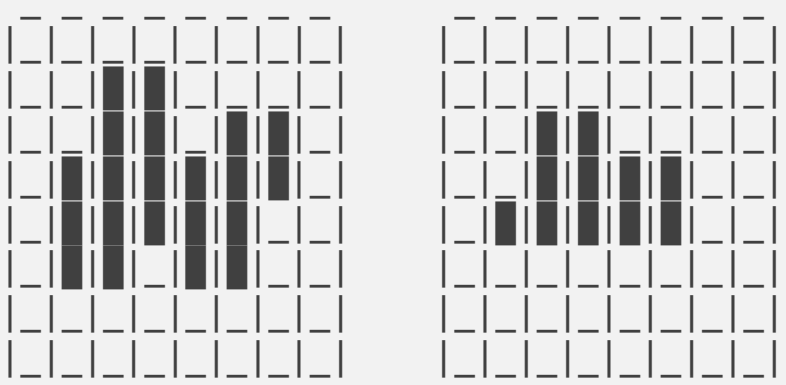
Предположение: 1234

Отговор: „1 бик и 2 крави“. (Бикът е „2“, а кравите са „4“ и „1“.)

На всеки ход играчите записват предположените числа и отговорите, за да могат чрез дедукция да идентифицират цифрите в тайното число на противника.

Problem: Melting Iceberg

Айсберг има форма, която може да се изобрази в таблица с N реда и N стълба, 7 < N < 200, например айсбергът от фиг. 1 след един час в резултат на топенето се превръща в айсберга от фиг. 2:



фиг. 1 фиг. 2

Клетките от първия и последния ред и стълб са винаги празни. Външните клетки, които са изложени на съприкосновение с топлия въздух и вода се топят, а вътрешните не. Айсбергът се топи по следното правило: всяка клетка която има поне 2 от съседните 4 клетки (с обща страна) празни се стопява изцяло за 1 час, а останалите клетки не се топят изобщо. Напишете програма, която прочита от текстов файл размера и съдържанието на таблицата:

```
8
00000000
00**0000
00**0**0
0******0
0*****00
0**0**00
00000000
00000000
```

В резултат програмата следва да извежда на екрана броя часове, за които айсбергът ще се разтопи изцяло. В горния пример изходът на програмата следва да бъде: 4.

Problem: Melting Iceberg II

Реализирайте конзолно приложение за интерактивно въвеждане и редактиране на таблицата от задача 3. Приложението следва да поддържа текстово меню с възможности за:

1) редактиране на таблицата;

2) създаване на нова празна таблица с възможност за редактиране;

3) прочитане на таблицата от текстов файл;

4) запис на таблицата в текстов файл;

5) изход от програмата.

Редактирането на таблицата трябва да стане в текстов вид, интерактивно от клавиатурата с поддържане на активен курсор (символ '#') и с натискане на '+' за запълване на клетката където е курсора и '-' за изчистване на клетката, където е курсора. Преместването на курсора става със стрелките от клавиатурата.

След натискане на всеки клавиш се извежда цялата таблица и един празен ред за разделител. Редактирането приключва с натискане на клавиша <Enter>, след което се връщаме в главното меню на програмата, като редактираната таблица се запомня.

Conclusion

- 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

Thank's for Your Attention!

Trayan Iliev

IPT – Intellectual Products & Technologies

http://iproduct.org/

https://github.com/iproduct

https://twitter.com/trayaniliev

https://www.facebook.com/IPT.EACAD