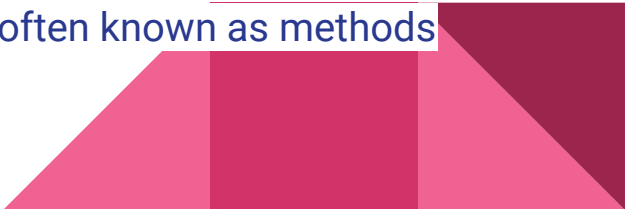


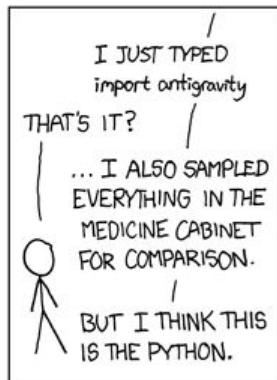
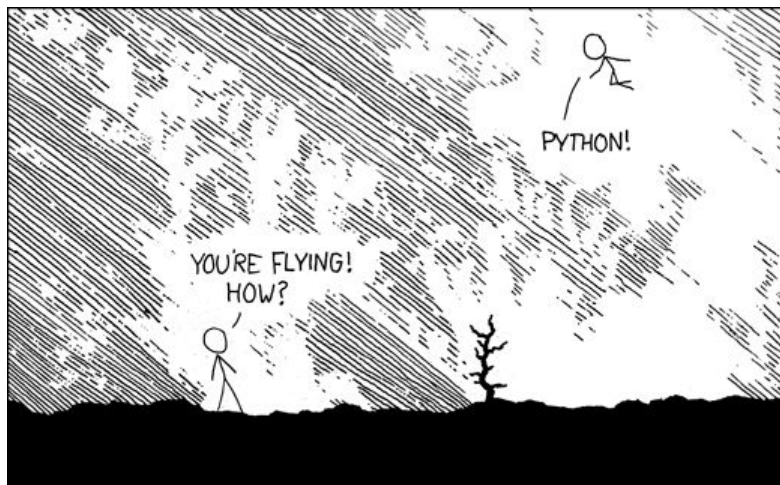
Python Data types

Petr Svarny, 2020

Type of programming languages

- Functional
 - Treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data
 - Elm, Haskell, Clojure, (Python)
 - Procedural
 - Based upon the concept of the procedure call (series of computational steps to be carried out)
 - BASIC, C, Python
 - Object-oriented
 - Based on the concept of “objects”, which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods
 - Ruby, Scala, Python
- 

Why Python?



Why Python?

- Easy readability (i.e. good scripting tool)
- Big community
- Diverse application (web, data science, machine learning)
- Used by leading companies (Google, Spotify, Dropbox, Disney, NASA)



Python in terminal

If you have more python versions (Mac OS, Linux)

python3

```
ja@muji-VirtualBox:~$ python3
Python 3.5.2 (default, Sep 10 2016, 08:21:44)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>> |
```

windows

python

```
MINGW32/c/Users/TEUser/Documents
TEUser@IE11Win7 MINGW32 ~/Documents
$ python
Python 3.6.1 |Anaconda custom (32-bit)| (default, May 11 2017, 14:16:49) [MSC v.
1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Starting Jupyter Notebook

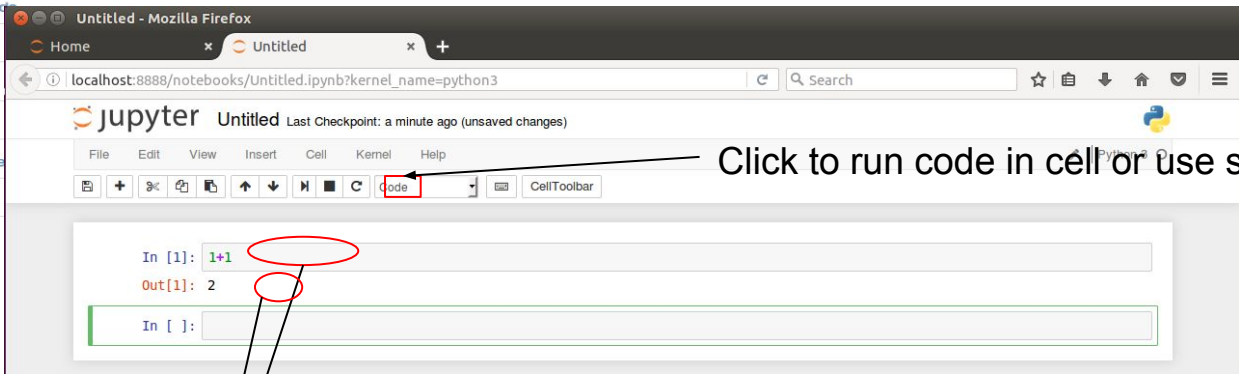
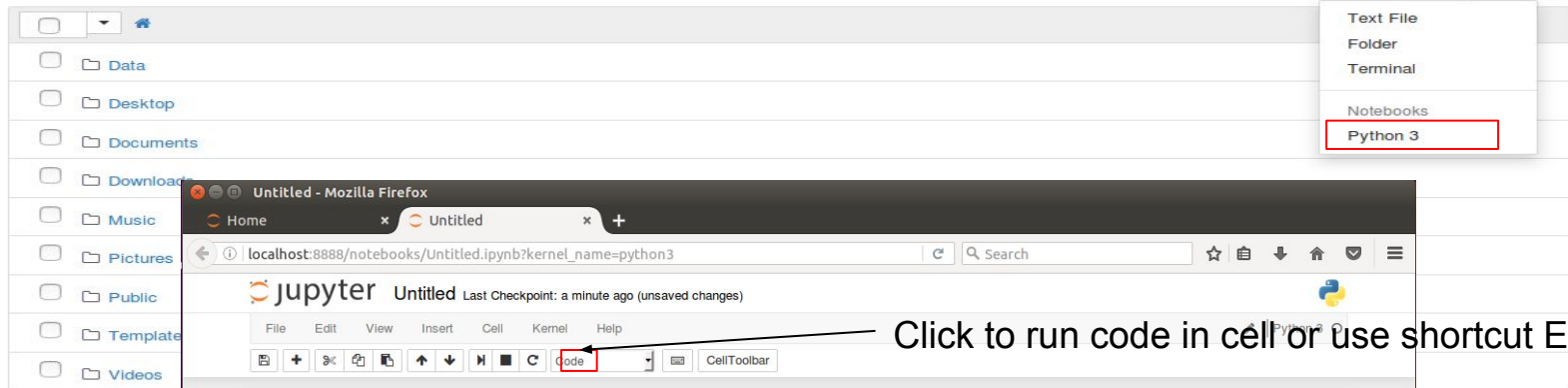
- In terminal
 - `jupyter notebook`
 - **Warning: you need to be in directory or in directory above where you want to create JupyterNotebooks**



Create Jupyter Python notebook

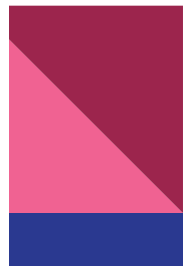


Files Running Clusters
Select items to perform actions on them.



Click to run code in cell or use shortcut Enter+Shift

```
>>> 1 + 1  
2
```



Python as calculator

```
>>> 1 + 1
```

```
2
```

```
>>> 5/2
```

```
2.5
```

```
>>> 3**2
```

```
9
```



Integer versus float

- Integer

```
>>> type(10)  
int
```

- Float

```
>>> type(10.0)  
float
```



Operations with int and float

- Modulo - remainder after division

```
>>> 10%5
```

```
0
```

```
>>> 10%3
```

```
1
```

- Whole number division

```
>>> 10//3
```

```
3
```



Exercise

- The king with lots of children got heritage - 1256983 golden coins. He decided that he will divide his heritage evenly among all of his 28 children. How many coins will remain to him?



Boolean value

- True, False

```
>>> 10/2 < 3
```

```
False
```

```
>>> 3**2 == 45/5
```

```
True
```

- True has value 1, False 0

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

```
True
```

```
>>> 0 == True
```

```
False
```



Relation operators

- $x == y$ *# x equals y*
 `>>> 3+7 == 5*2`
 True
- $x != y$ *# x not equals y*
 `>>> 3**2 != 3**3`
 True
- $x > y$ *# x more than y*
 `>>> 15 > 10`
 True



Relation operators

- $x < y$ *# x less than y*
 `>>> 100 < 1000`
 True
- $x \geq y$ *# x more or equal than y*
 `>>> 10 \geq 9`
 True
- $x \leq y$ *# x less or equal than y*
 `>>> 1 \leq 2`
 True



Logic operators

- And

```
>>> 2*4 == 8 and 15-7 == 8
```

```
True
```

- Or

```
>>> 2*4 == 8 or 15-10 == 8
```

```
True
```

- Not

```
>>> not (1+1 == 3)
```

```
True
```



Exercise

- Evaluate if following is true
 - Remainder after division of 12^{52} by 15 is less than 8 or 3^5 is more than 100
 - 5 times 3^3 is not equal to 900 divided by 75

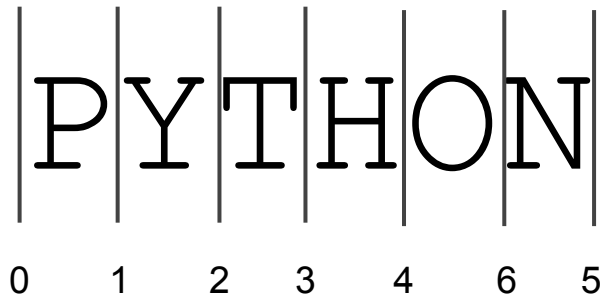


String

- Complex data type
 - Contains smaller parts (numbers, letter, etc.)
- Is 0-based indexed

```
>>> 'PYTHON'[0]  
'P'
```

| P | Y | T | H | O | N |
0 1 2 3 4 6 5



String

- One line

```
>>> type('Anna')
```

```
str
```

```
>>> type("Anna")
```

```
str
```

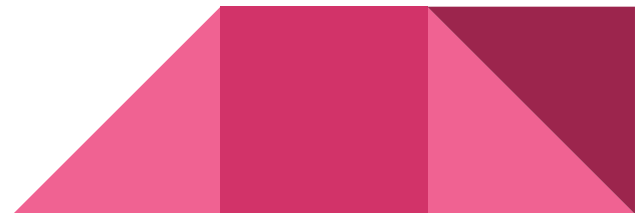
- Multiple lines

```
>>> """ Very long
```

```
... sentence
```

```
... """
```

```
' Very long\nsentence\n'
```



String operations

- Get string's length

```
>>> len('python')
```

```
6
```

- Select substring [start: end: step]

- Default values [0: string length: 1]

```
>>> 'python'[::]
```

```
'python'
```

```
>>> 'python'[1::2]
```

```
'yhn'
```

```
>>> 'python'[-1]
```

```
'n'
```



String operations

- Adding

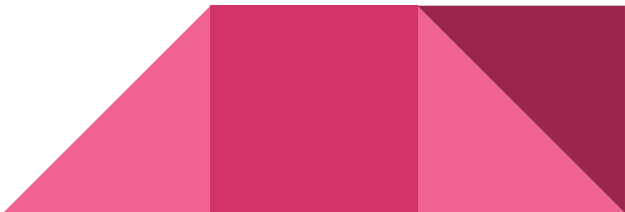
```
>>> 'My name is ' + 'Anna'  
'My name is Anna'
```

- Repetition

```
>>> 'Hello' * 3  
'HelloHelloHello'
```

- Comparing

```
>>> 'Almond'[0] == 'Apple'[0]  
True
```



Exercise

- Create one string from the 2 following strings:
 - '[' + 'PYTHON' + ']' → '[[PYTHON]]'

- Create one string from another
 - 'Python' → 'onononon'
 - 'Perl' → 'rrrrrr'



String operations - methods

- Convert all letters lower case - `str.lower()`, opposite method- `str.upper()`

```
>>> 'I FEEL GREAT'.lower()
```

```
'i feel great'
```

- Capitalize first letter in string - `str.capitalize()`

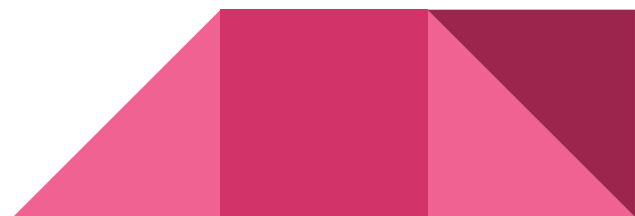
```
>>> 'prague is beautiful!.capitalize()
```

```
'Prague is beautiful!'
```

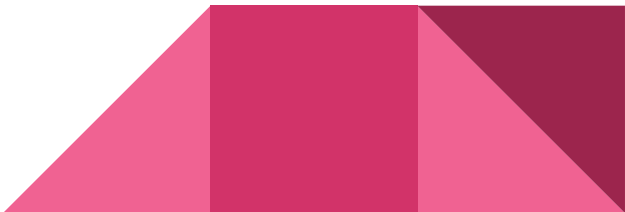
- Count substring occurrence in string - `str.count()`

```
>>> 'Long long long day'.count('long')
```

```
3
```



Functions and methods

- All methods are functions, but not all functions are methods
 - Methods are functions for specific class of objects (e.g. strings)
 - Call function
 - `function(parameters)`
 - `len('python')`
 - Call methods
 - `object.methods(parameters)`
 - `'python'.count('n')`
- 

Exercise

- Make first half of the string uppercase and second half lowercase
 - E.g. 'python' ---> 'PYThon'

- Create string which is first letter of another string repeated n number of times, where n - length of the given string
 - E.g. 'python' ---> 'ppppppp'
 - 'git' ---> 'ggg'



String operations

- Search in string

```
>>> 'G' in 'AACCTCA'
```

```
False
```

- And much more...

```
>>> dir(str)
```

```
[..., 'capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs',  
'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',  
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper',  
'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex',  
'rjust', 'rpartition', ...]
```

Change data type

```
>>> int(10.0)
```

```
10
```

```
>>> float(10)
```

```
10.0
```

```
>>> str(10)
```

```
'10'
```



Exercise

- What results do you get when you run following expressions?
 - `print(7+3*2)`
 - `print('7' + str(3*2))`
 - `print('7' + '3*2')`
 - `print('7' + 3*2)`
- What is causing error ?



Variables

- Assign variable using equals sign

```
>>> my_name = 'Anna'
```

```
>>> my_name
```

```
'Anna'
```

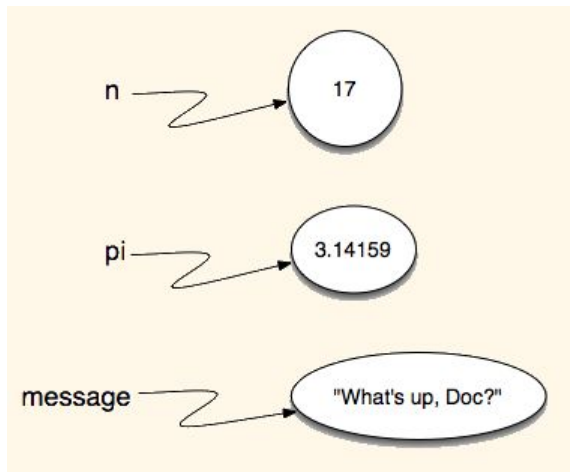
```
>>> My_name
```

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

```
File "<ipython-input-59-9804048db7e1>", line 1, in <module>
```

```
Name
```

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



How to name variables in Python?

- In English

👍 name

👎 nombre

- Underscore to divide words (variables and functions)

👍 friend_name

👎 friendname friendName

- Help to decipher variables for someone, who will read the code after you

👍 selected_color

👎 my_variable

- Not "0", "l", "1"

Format method

- Replace variable parts in string
- More info and examples: <https://pyformat.info/>

```
>>>name = 'Anna'
```

```
>>>color = 'blue'
```

```
>>>'Hello, my name is {0} and my favourite color  
is {1}.'.format(name, color)
```

```
'Hello, my name is 3.141592653589793 and my favourite color  
is blue.'
```

```
'{:0.2f}'.format(3.141592653589793)
```

```
'3.14'
```

Exercise

- Save your hobby to the 'hobby' variable
 - Using `format` method print following string 'My hobby is <your hobby>.' with completed hobby
- Transform variable `date = '2018-11-01'` to `'01/11'` using `format`



List

- Complex data type
 - Can contain strings, numbers, other lists, etc.
- 0-based indexing, the same as for string

```
>>> my_list = ['apple', 42, 'John', ['a', 'b']]
```

```
>>> type(my_list)
```

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

```
>>> nucl = 'Anna'
```

```
>>> list(nucl)
```

```
['A', 'n', 'n', 'a']
```


List

- Lists are *mutable*
 - Unlike strings, elements of list can be changed

```
>>> my_list[3] = 'banana'  
>>> my_list  
['apple', 42, 'John', 'banana']
```

```
>>> name = 'Anna'  
>>> name[0] = 'C'
```

Traceback (most recent call last):

File "<ipython-input-9-69728e5ccb1f>", line 1, in <module>

```
name[0] = 'C'
```

TypeError: 'str' object does not support item assignment

List operations

- Get length

```
>>> numbers = ['one', 'two', 'three', 'four']
```

```
>>> len(numbers)
```

```
4
```

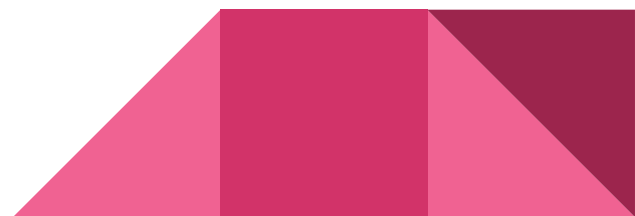
- Subset

```
>>> numbers[0::2]
```

```
['one', 'three']
```

```
>>> numbers[-1]
```

```
'four'
```



List operations

- Add lists

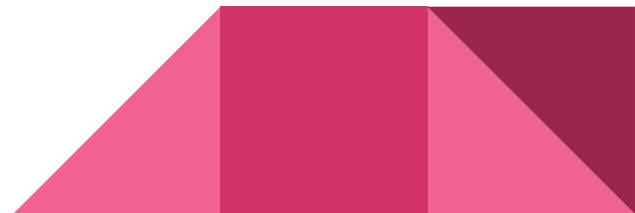
```
>>> my_list + numbers  
['banana', 42, 'John', 'apple', 'one', 'two', 'three', 'four']
```

- Multiply lists

```
>>> ['Hello']*3  
['Hello', 'Hello', 'Hello']
```

- Append elements

```
>>> numbers.append('five')  
>>> numbers  
['one', 'two', 'three', 'four', 'five']
```



List operations

- Delete elements

```
>>> del numbers[1]
>>> numbers
['one', 'three', 'four', 'five']
```

- Sort elements

```
>>> fruits = ['banana', 'apple', 'mango', 'kiwi']
>>> fruits.sort()
>>> print(fruits)
['apple', 'banana', 'kiwi', 'mango']
```

- Other list methods

```
>>> dir(list)
[... , 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```




Exercise

- Create list of your hobbies by adding elements to the empty list (the favourite one will be the first)
- Print the hobby you like most
- Print the hobby you like least
- Delete it



Exercise

- List of the Czech cities sorted by population size
 - `cities = ['Prague', 'Brno', 'Ostrava', 'Plzen', 'Liberec', 'Olomouc', 'Usti nad Labem', 'Hradec Kralove', 'Ceske Budejovice', 'Pardubice']`
 - Sort cities in alphabetical order
 - Using `"".join()` join cities into one string separated by asterix (*)
- 

Set

- Unordered mutable collection of unique elements

- To create use curly braces

```
>>> basket = {'apple','orange','apple','pear','orange','banana'}
```

```
>>> basket
```

```
{'apple', 'banana', 'orange', 'pear'}
```

```
>>> type(basket)
```

```
<class 'set'>
```

- Or set() function

```
>>> set('Anna')
```

```
{'A', 'a', 'n'}
```

Set

```
>>> basket = {'apple','orange','apple','pear','orange','banana'}  
>>> basket  
{'apple', 'banana', 'orange', 'pear'}
```

- Set is not indexed

```
>>> basket[0]
```

Traceback (most recent call last):

File "<ipython-input-7-0ed6bd42dd11>", line 1, in <module>

```
basket[0]
```

```
TypeError: 'set' object does not support indexing
```


Set

- Basic uses of set include membership testing and eliminating duplicate entries

```
>>> my_friends = set(['Emily', 'Hannah', 'John'])
```

```
>>> your_friends = set(['James', 'Carol', 'Hannah'])
```

```
>>> my_friends.intersection(your_friends)
```

```
{'Hannah'}
```

```
>>> my_friends.difference(your_friends)
```

```
{'Emily', 'John'}
```

```
>>>
```



Tuple

- Similar to list, but it is immutable

```
>>> my_tuple = ('a', 'b', 'c', 'd')
```

```
>>> type(my_tuple)
```

```
<class 'tuple'>
```

```
>>> my_tuple[0] = 'z'
```

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

```
File "<ipython-input-4-75e90a118977>", line 1, in <module>
```

```
my_tuple[0] = 'z'
```

```
TypeError: 'tuple' object does not support item assignment
```

Dictionary

- Complex data type, each element contain key:value pair

```
>>> info = {'name':'Anna', 'dog': 'Miky'}  
>>> type(info)  
dict
```

- Key can be any *immutable* data type, value can be any data type

```
>>> info = [{'name', 'first'}: 'Anna']  
Traceback (most recent call last):  
File "<ipython-input-16-7ab66cffb746>", line 1, in <module>  
info = [{'name', 'first'}: 'Anna']  
TypeError: unhashable type: 'list'
```

Accessing elements of dictionary

- dict['key']

```
>>> info['name']
```

```
'Anna'
```

```
>>> info.values()
```

```
dict_values(['Miky', 'Anna'])
```

```
>>> info.keys()
```

```
dict_keys(['dog', 'name'])
```

- Keys can not be duplicate

```
>>> info = {'name': 'Anna', 'dog': 'Miky', 'name': 'Julie', 'dog': 'Rex'}
```

```
>>> info['name']
```

```
'Julie'
```

Dictionary operations

- Assign value to key

```
>>> info['color'] = 'blue'
```

```
>>> info
```

```
{'color': 'blue', 'dog': 'Miky', 'name': 'Nasta'}
```

- Get key-value pairs

```
>>> info.items()
```

```
dict_items([('color', 'blue'), ('dog', 'Miky'), ('name', 'Nasta')])
```



Dictionary operations

- Get keys

```
>>> info.keys()
dict_keys({'color', 'dog', 'name'})
```

- Get values

```
>>> info.values()
dict_values(['blue', 'Miky', 'Nasta'])
```



Dictionary operations

- Delete key-value pair

```
>>>del info['name']
```

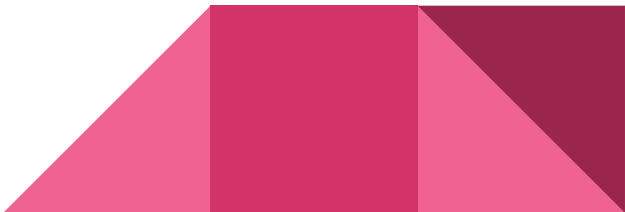
```
>>>info
```

```
{'color': 'blue', 'dog': 'Miky'}
```

- Other methods

```
>>> dir(dict)
```

```
[..., 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem',  
'setdefault', 'update', 'values']
```



How to create...

- **String**
 - `name = 'apple'`
- **List**
 - `stuff = [42, 'dog', ['drawing', 'running', 'reading']]`
- **Set**
 - `fruits = {'apples', 'banana', 'pears'}`
- **Dictionary**
 - `phonebook = {'Lenka': 700523698, 'Jitka': 563412789}`
- **Tuple**
 - `coords = (25, 64)`

Exercise

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'  
zen = """
```

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.

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!

```
"""
```

- Find what letters are not present in the Zen of Python
- Hint: use `set()`



Exercise

- Fix dictionary and delete erroneous key
 - `d = {'python':'An interpreted, object-oriented programming language'}`
- Use one of the data types and create dictionary, where key will be name and surname (do not put it at one string) and value will be telephone number



Exercise

- You have a dictionary
 - `info = {'Name', 'Surname':('John', 'Doe')}`
- Extract dictionary value in following format
 - `'John_Doe'`



Where to search for help by yourself?

- [Documentation](#)
- [Stackoverflow.com](#)
- [Python community](#)

