Data Science

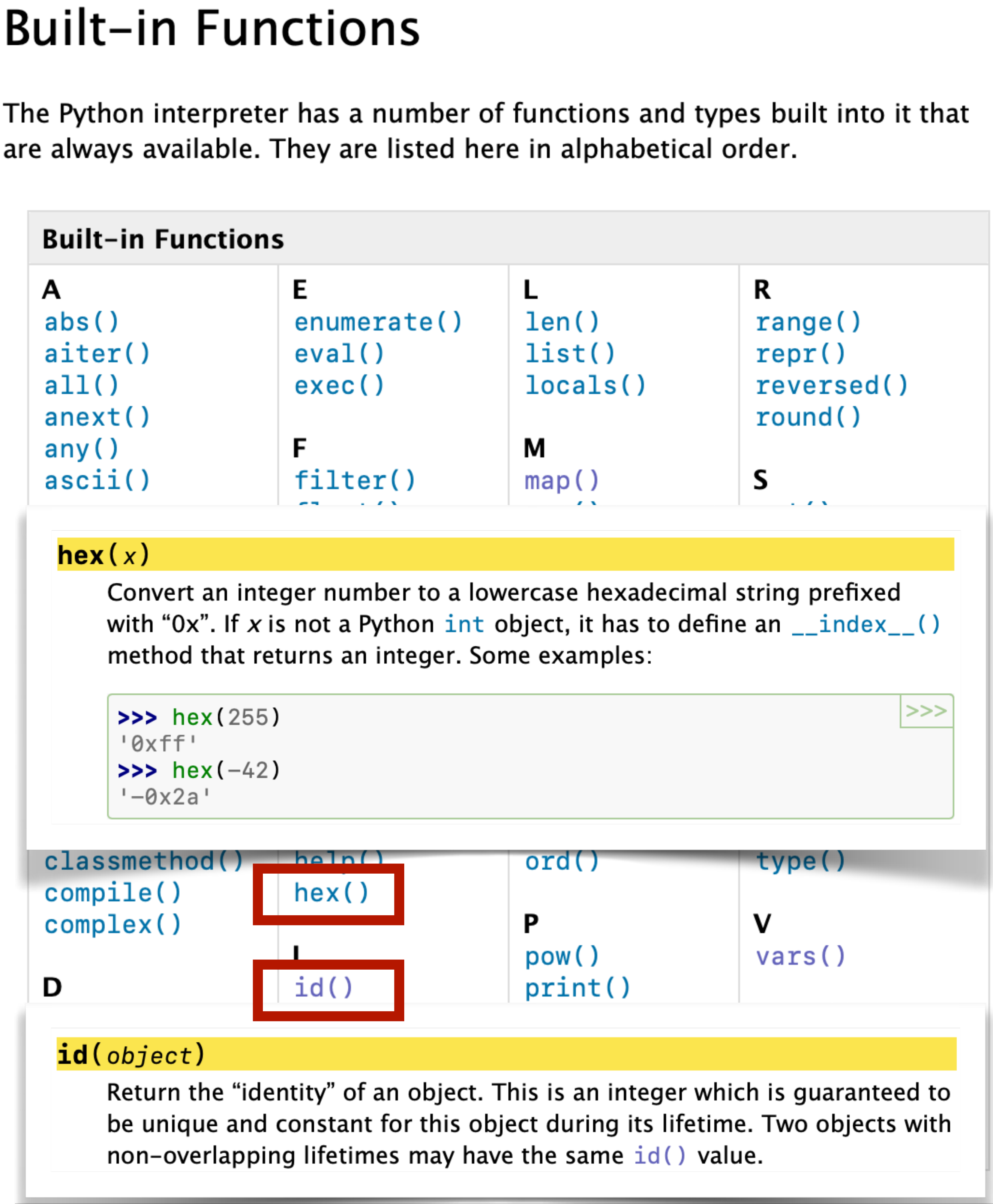
Assoc. Prof. Dr. Bora Canbula

https://github.com/canbula/DataScience/

uh6fveq

Variables

Variables are symbols for memory addresses.



Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions

A
abs()
aiter()
all()
anext()
any()
ascii()

E
enumerate()
eval()
exec()

F
filter()

L
len()
list()
locals()

M
map()

R
range()
repr()
reversed()
round()

S

hex(*x*)

Convert an integer number to a lowercase hexadecimal string prefixed with "0x". If *x* is not a Python int object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> hex(255)
'0xff'
>>> hex(-42)
'-0x2a'
```

classmethod()
compile()
complex()

D

help()
hex()

I
id()

ord()

P
pow()
print()

type()

V
vars()

id(*object*)

Return the "identity" of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same `id()` value.

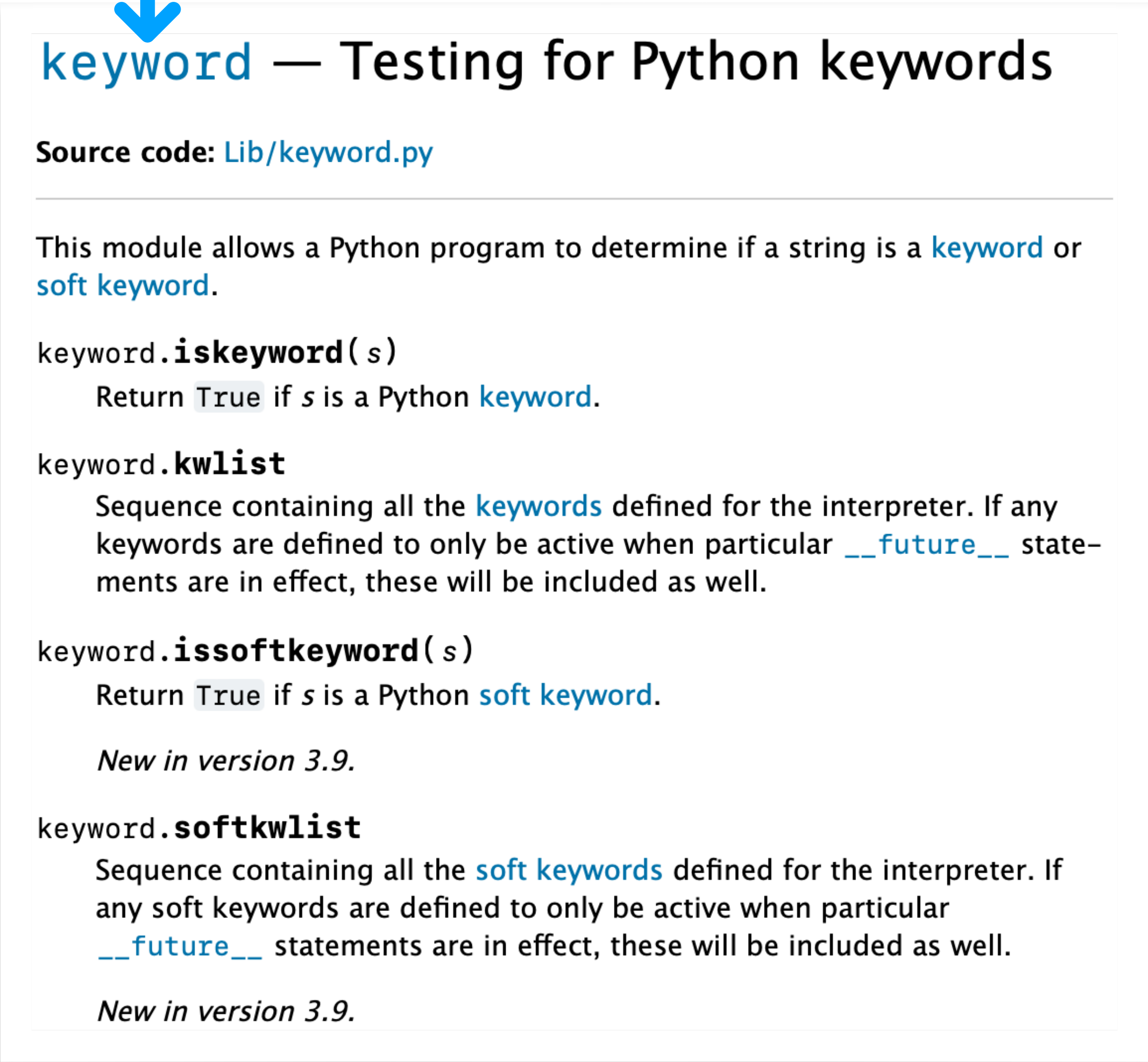
https://docs.python.org/3/library/functions.html

Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

Rules

- Names are case sensitive.
- Names can be a combination of letters, digits, and underscore.
- Names can only start with a letter or underscore, can not start with a digit.
- Keywords can not be used as a name.



keyword — Testing for Python keywords

Source code: Lib/keyword.py

This module allows a Python program to determine if a string is a keyword or soft keyword.

`keyword.`**`iskeyword`**`(`*s*`)`

Return `True` if *s* is a Python keyword.

`keyword.`**`kwlist`**

Sequence containing all the keywords defined for the interpreter. If any keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

`keyword.`**`issoftkeyword`**`(`*s*`)`

Return `True` if *s* is a Python soft keyword.

New in version 3.9.

`keyword.`**`softkwlist`**

Sequence containing all the soft keywords defined for the interpreter. If any soft keywords are defined to only be active when particular `__future__` statements are in effect, these will be included as well.

New in version 3.9.

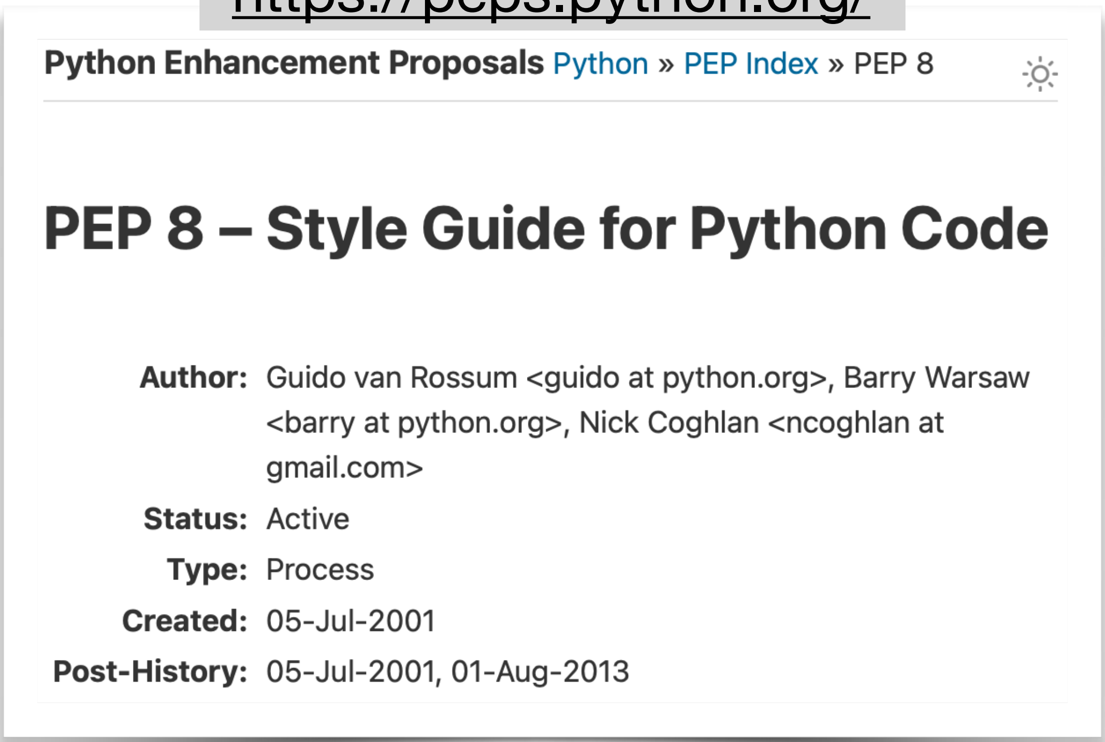
Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

Rules

- Names are case sensitive.
- Names can be a combination of letters, digits, and underscore.
- Names can only start with a letter or underscore, can not start with a digit.
- Keywords can not be used as a name.

https://peps.python.org/



Python Enhancement Proposals Python » PEP Index » PEP 8

PEP 8 – Style Guide for Python Code

Author:	Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Nick Coghlan <ncoghlan at gmail.com>
Status:	Active
Type:	Process
Created:	05-Jul-2001
Post-History:	05-Jul-2001, 01-Aug-2013

Identifier Names

For variables, functions, classes etc. we use identifier names. We must obey some rules and we should follow some naming conventions.

Conventions

- Names to Avoid
  Never use the characters ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), or ‘I’ (uppercase letter eye) as single character variable names.
- Packages
  Short, all-lowercase names without underscores
- Modules
  Short, all-lowercase names, can have underscores
- Classes
  CapWords (upper camel case) convention
- Functions
  snake_case convention
- Variables
  snake_case convention
- Constants
  ALL_UPPERCASE, words separated by underscores

Leading and Trailing Underscores

- `_single_leading_underscore`
  Weak “internal use” indicator.
  `from M import *` does not import objects whose names start with an underscore.
- `single_trailing_underscore_`
  Used by convention to avoid conflicts with keyword.
- `__double_leading_underscore`
  When naming a class attribute, invokes name mangling (inside class `FooBar`, `__boo` becomes `_FooBar__boo`)
- `__double_leading_and_trailing_underscore__`
  “magic” objects or attributes that live in user-controlled namespaces (`__init__`, `__import__`, etc.). Never invent such names; only use them as documented.

Variable Types

Python is dynamically typed. Python does not have primitive types. Everything is an object in Python, therefore, a variable is purely a reference to an object with the specified value.

Numeric Types

- Integer
- Float
- Complex
- Boolean

Formatted Output

```
print(“static text = ”, variable)
print(“static text = %d” % (variable))
print(“static text = {0}”.format(variable))
print(f”static text = {variable}”)
print(f”static text = {variable:5d}”)
```

Variable Types

Python is dynamically typed. Python does not have primitive types. Everything is an object in Python, therefore, a variable is purely a reference to an object with the specified value.

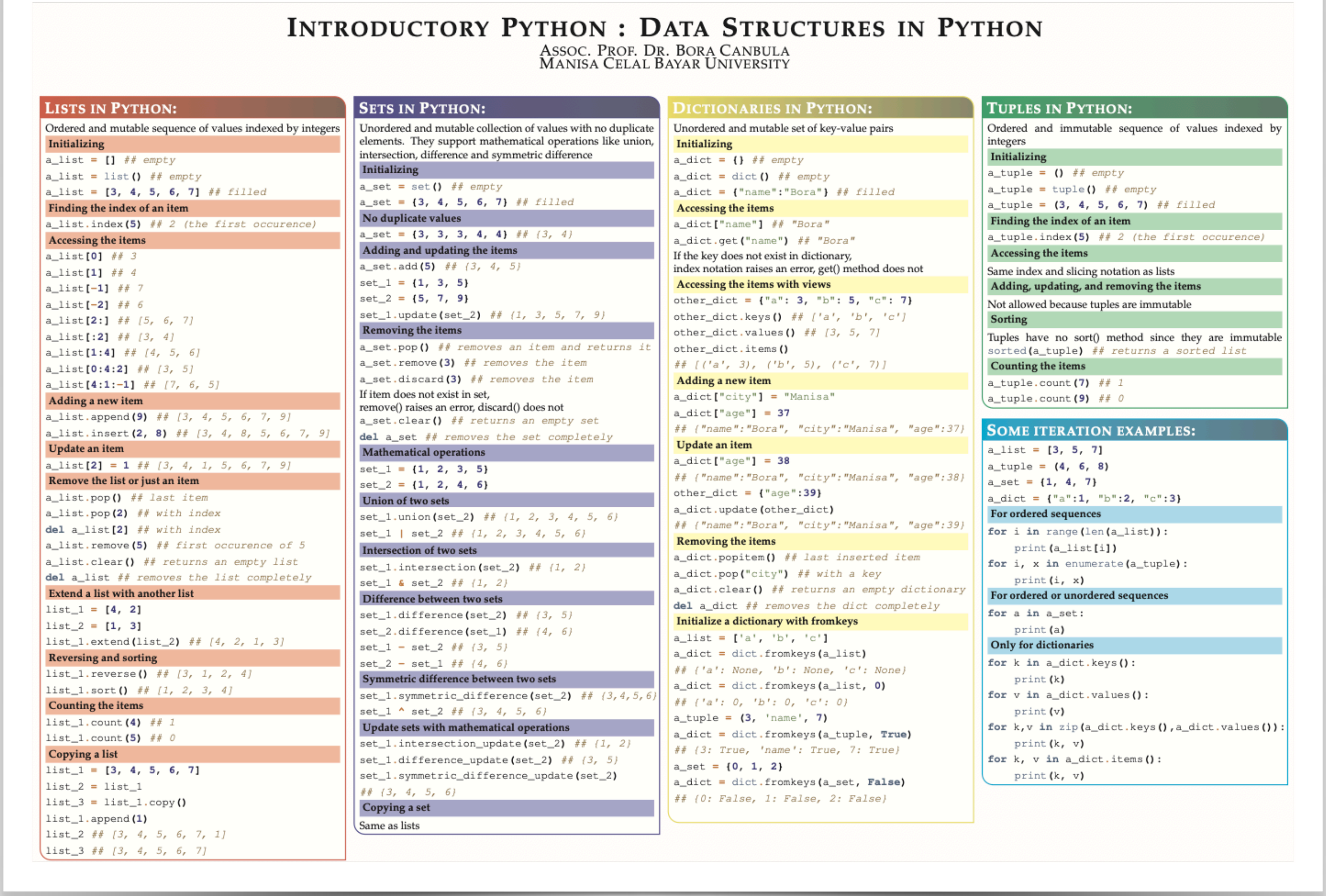
Numeric Types

- Integer
- Float
- Complex
- Boolean

Sequences

- Strings
- List
- Tuple
- Set
- Dictionary

Week02/IntroductoryPythonDataStructures.pdf



Introductory Python : Data Structures in Python

Assoc. Prof. Dr. Bora Canbula
Manisa Celal Bayar University

Lists in Python:

Ordered and mutable sequence of values indexed by integers

Initializing

```
a_list = [] ## empty
a_list = list() ## empty
a_list = [3, 4, 5, 6, 7] ## filled
```

Finding the index of an item

```
a_list.index(5) ## 2 (the first occurence)
```

Accessing the items

```
a_list[0] ## 3
a_list[1] ## 4
a_list[-1] ## 7
a_list[-2] ## 6
a_list[2:] ## [5, 6, 7]
a_list[:2] ## [3, 4]
a_list[1:4] ## [4, 5, 6]
a_list[0:4:2] ## [3, 5]
a_list[4:1:-1] ## [7, 6, 5]
```

Adding a new item

```
a_list.append(9) ## [3, 4, 5, 6, 7, 9]
a_list.insert(2, 8) ## [3, 4, 8, 5, 6, 7, 9]
```

Update an item

```
a_list[2] = 1 ## [3, 4, 1, 5, 6, 7, 9]
```

Remove the list or just an item

```
a_list.pop() ## last item
a_list.pop(2) ## with index
del a_list[2] ## with index
a_list.remove(5) ## first occurence of 5
a_list.clear() ## returns an empty list
del a_list ## removes the list completely
```

Extend a list with another list

```
list_1 = [4, 2]
list_2 = [1, 3]
list_1.extend(list_2) ## [4, 2, 1, 3]
```

Reversing and sorting

```
list_1.reverse() ## [3, 1, 2, 4]
list_1.sort() ## [1, 2, 3, 4]
```

Counting the items

```
list_1.count(4) ## 1
list_1.count(5) ## 0
```

Copying a list

```
list_1 = [3, 4, 5, 6, 7]
list_2 = list_1
list_3 = list_1.copy()
list_1.append(1)
list_2 ## [3, 4, 5, 6, 7, 1]
list_3 ## [3, 4, 5, 6, 7]
```

Sets in Python:

Unordered and mutable collection of values with no duplicate elements. They support mathematical operations like union, intersection, difference and symmetric difference

Initializing

```
a_set = set() ## empty
a_set = {3, 4, 5, 6, 7} ## filled
```

No duplicate values

```
a_set = {3, 3, 3, 4, 4} ## {3, 4}
```

Adding and updating the items

```
a_set.add(5) ## {3, 4, 5}
set_1 = {1, 3, 5}
set_2 = {5, 7, 9}
set_1.update(set_2) ## {1, 3, 5, 7, 9}
```

Removing the items

```
a_set.pop() ## removes an item and returns it
a_set.remove(3) ## removes the item
a_set.discard(3) ## removes the item
```

If item does not exist in set,
remove() raises an error, discard() does not

```
a_set.clear() ## returns an empty set
del a_set ## removes the set completely
```

Mathematical operations

```
set_1 = {1, 2, 3, 5}
set_2 = {1, 2, 4, 6}
```

Union of two sets

```
set_1.union(set_2) ## {1, 2, 3, 4, 5, 6}
set_1 | set_2 ## {1, 2, 3, 4, 5, 6}
```

Intersection of two sets

```
set_1.intersection(set_2) ## {1, 2}
set_1 & set_2 ## {1, 2}
```

Difference between two sets

```
set_1.difference(set_2) ## {3, 5}
set_2.difference(set_1) ## {4, 6}
set_1 - set_2 ## {3, 5}
set_2 - set_1 ## {4, 6}
```

Symmetric difference between two sets

```
set_1.symmetric_difference(set_2) ## {3,4,5,6}
set_1 ^ set_2 ## {3, 4, 5, 6}
```

Update sets with mathematical operations

```
set_1.intersection_update(set_2) ## {1, 2}
set_1.difference_update(set_2) ## {3, 5}
set_1.symmetric_difference_update(set_2)
## {3, 4, 5, 6}
```

Copying a set

Same as lists

Dictionaries in Python:

Unordered and mutable set of key-value pairs

Initializing

```
a_dict = {} ## empty
a_dict = dict() ## empty
a_dict = {"name":"Bora"} ## filled
```

Accessing the items

```
a_dict["name"] ## "Bora"
a_dict.get("name") ## "Bora"
```

If the key does not exist in dictionary,
index notation raises an error, get() method does not

Accessing the items with views

```
other_dict = {"a": 3, "b": 5, "c": 7}
other_dict.keys() ## ['a', 'b', 'c']
other_dict.values() ## [3, 5, 7]
other_dict.items()
## [('a', 3), ('b', 5), ('c', 7)]
```

Adding a new item

```
a_dict["city"] = "Manisa"
a_dict["age"] = 37
## {"name":"Bora", "city":"Manisa", "age":37}
```

Update an item

```
a_dict["age"] = 38
## {"name":"Bora", "city":"Manisa", "age":38}
other_dict = {"age":39}
a_dict.update(other_dict)
## {"name":"Bora", "city":"Manisa", "age":39}
```

Removing the items

```
a_dict.popitem() ## last inserted item
a_dict.pop("city") ## with a key
a_dict.clear() ## returns an empty dictionary
del a_dict ## removes the dict completely
```

Initialize a dictionary with fromkeys

```
a_list = ['a', 'b', 'c']
a_dict = dict.fromkeys(a_list)
## {'a': None, 'b': None, 'c': None}
a_dict = dict.fromkeys(a_list, 0)
## {'a': 0, 'b': 0, 'c': 0}
a_tuple = (3, 'name', 7)
a_dict = dict.fromkeys(a_tuple, True)
## {3: True, 'name': True, 7: True}
a_set = {0, 1, 2}
a_dict = dict.fromkeys(a_set, False)
## {0: False, 1: False, 2: False}
```

Tuples in Python:

Ordered and immutable sequence of values indexed by integers

Initializing

```
a_tuple = () ## empty
a_tuple = tuple() ## empty
a_tuple = (3, 4, 5, 6, 7) ## filled
```

Finding the index of an item

```
a_tuple.index(5) ## 2 (the first occurence)
```

Accessing the items

Same index and slicing notation as lists

Adding, updating, and removing the items

Not allowed because tuples are immutable

Sorting

Tuples have no sort() method since they are immutable

```
sorted(a_tuple) ## returns a sorted list
```

Counting the items

```
a_tuple.count(7) ## 1
a_tuple.count(9) ## 0
```

Some iteration examples:

```
a_list = [3, 5, 7]
a_tuple = (4, 6, 8)
a_set = {1, 4, 7}
a_dict = {"a":1, "b":2, "c":3}
```

For ordered sequences

```
for i in range(len(a_list)):
    print(a_list[i])
for i, x in enumerate(a_tuple):
    print(i, x)
```

For ordered or unordered sequences

```
for a in a_set:
    print(a)
```

Only for dictionaries

```
for k in a_dict.keys():
    print(k)
for v in a_dict.values():
    print(v)
for k,v in zip(a_dict.keys(),a_dict.values()):
    print(k, v)
for k, v in a_dict.items():
    print(k, v)
```

Variable Types

Python is <u>dynamically typed</u>. Python does not have primitive types. Everything is an object in Python, therefore, a variable is purely a <u>reference to an object</u> with the specified value.

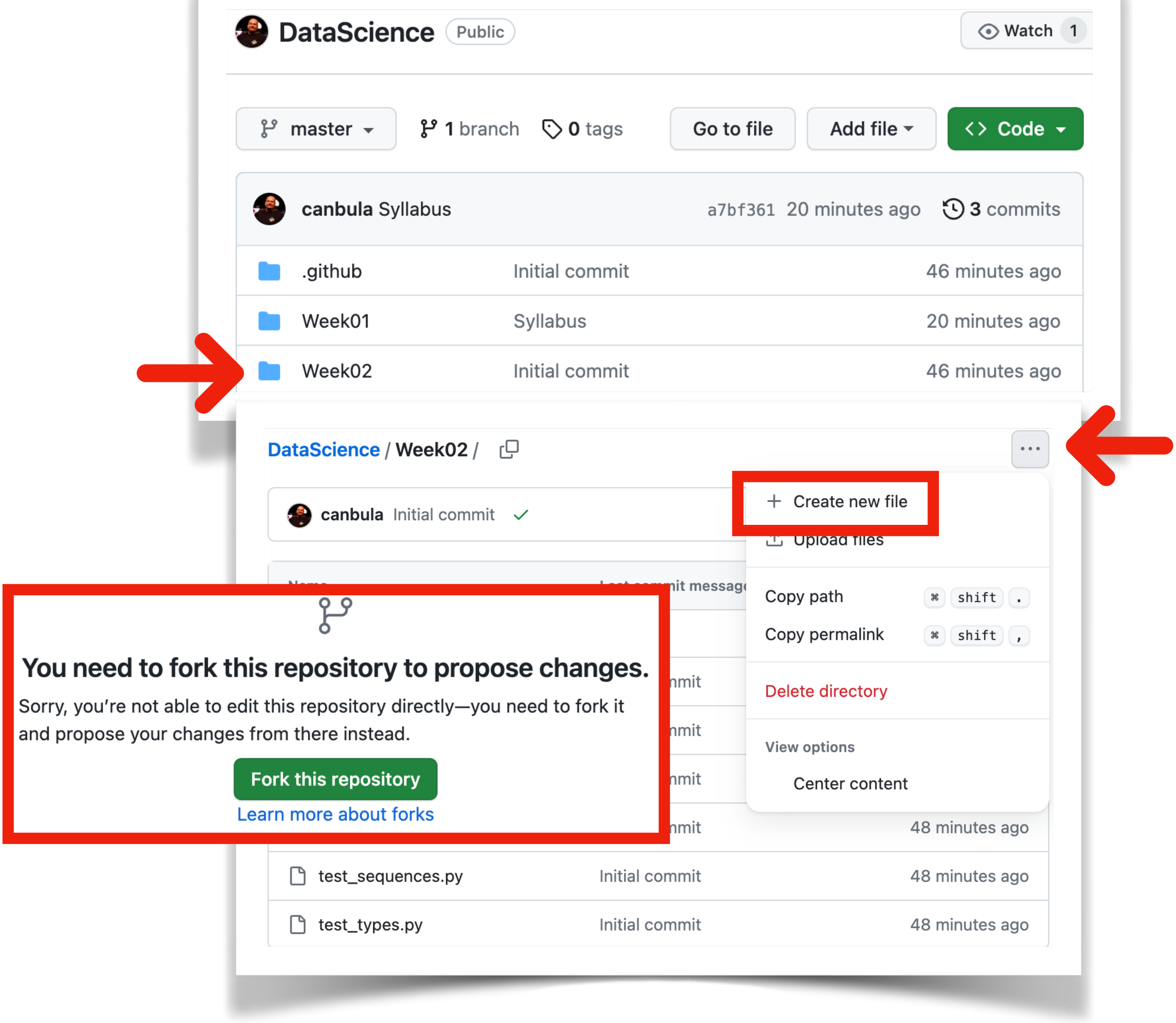
Numeric Types

- Integer
- Float
- Complex
- Boolean

Sequences

- Strings
- List
- Tuple
- Set
- Dictionary

Your First Homework



canbula Syllabus	a7bf361 20 minutes ago	3 commits
.github	Initial commit	46 minutes ago
Week01	Syllabus	20 minutes ago
Week02	Initial commit	46 minutes ago

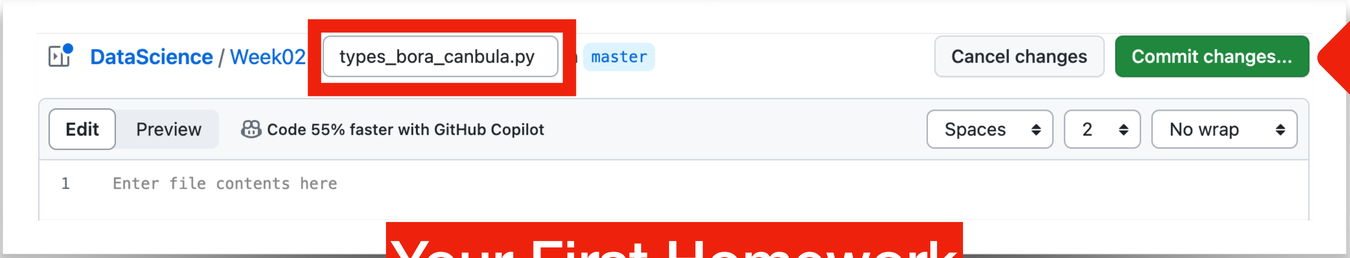
DataScience / Week02 /

canbula Initial commit ✓

Name	Last commit message	
	…mit	
	…mit	
	…mit	
	…mit	48 minutes ago
test_sequences.py	Initial commit	48 minutes ago
test_types.py	Initial commit	48 minutes ago

You need to fork this repository to propose changes.

Sorry, you're not able to edit this repository directly—you need to fork it and propose your changes from there instead.

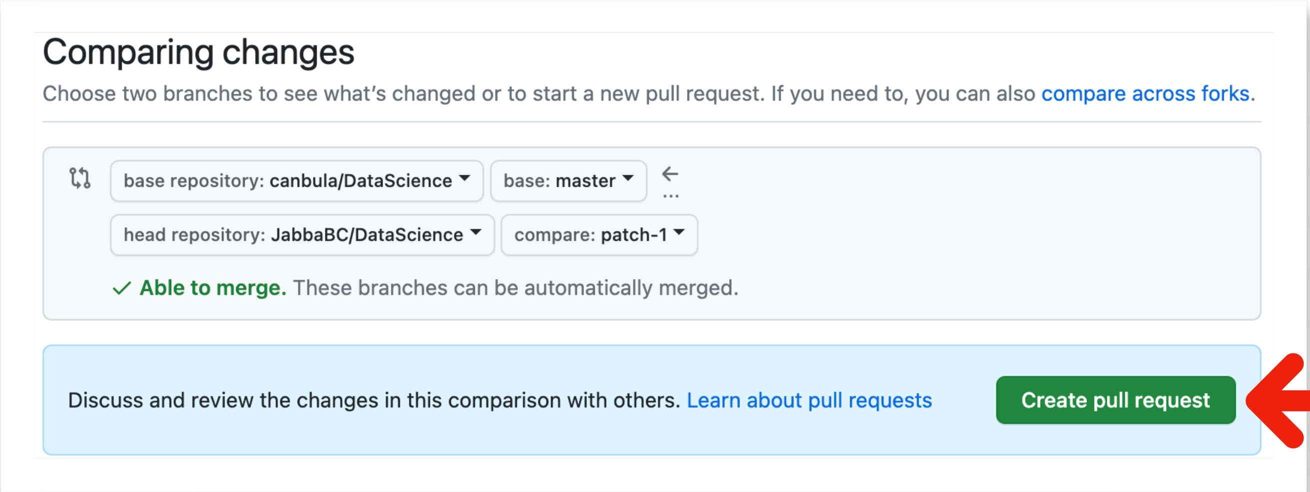


DataScience / Week02 / types_bora_canbula.py master

```
1   Enter file contents here
```

Your First Homework

- ☑ An integer with the name: `my_int`
- ☑ A float with the name: `my_float`
- ☑ A boolean with the name: `my_bool`
- ☑ A complex with the name: `my_complex`



Comparing changes

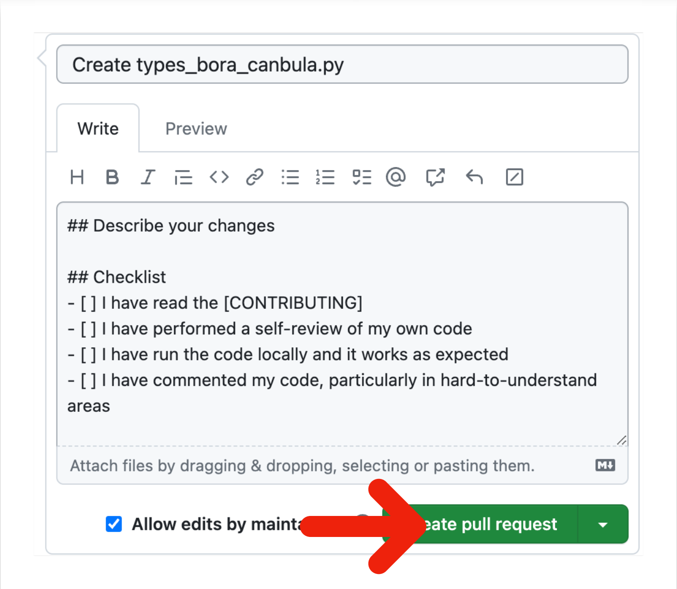
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks.

base repository: canbula/DataScience ▾ base: master ▾

head repository: JabbaBC/DataScience ▾ compare: patch-1 ▾

✓ Able to merge. These branches can be automatically merged.

Discuss and review the changes in this comparison with others. Learn about pull requests



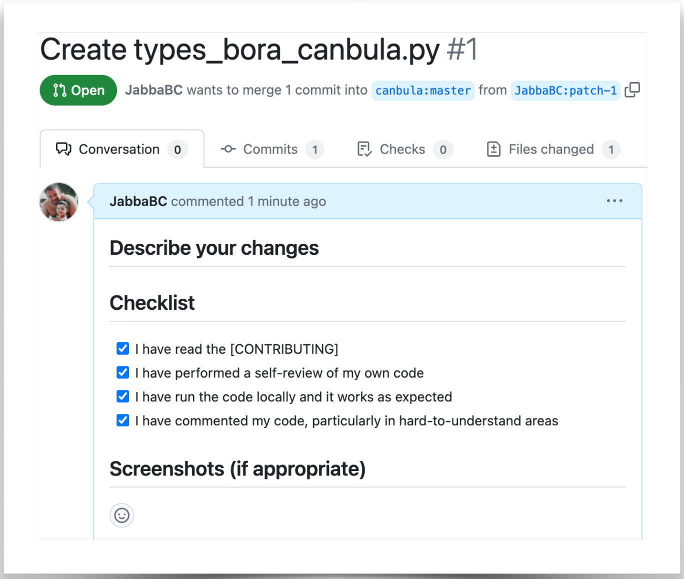
Create types_bora_canbula.py

```
## Describe your changes

## Checklist
- [ ] I have read the [CONTRIBUTING]
- [ ] I have performed a self-review of my own code
- [ ] I have run the code locally and it works as expected
- [ ] I have commented my code, particularly in hard-to-understand areas
```

Attach files by dragging & dropping, selecting or pasting them.

☑ Allow edits by maint



Create types_bora_canbula.py #1

Open JabbaBC wants to merge 1 commit into canbula:master from JabbaBC:patch-1

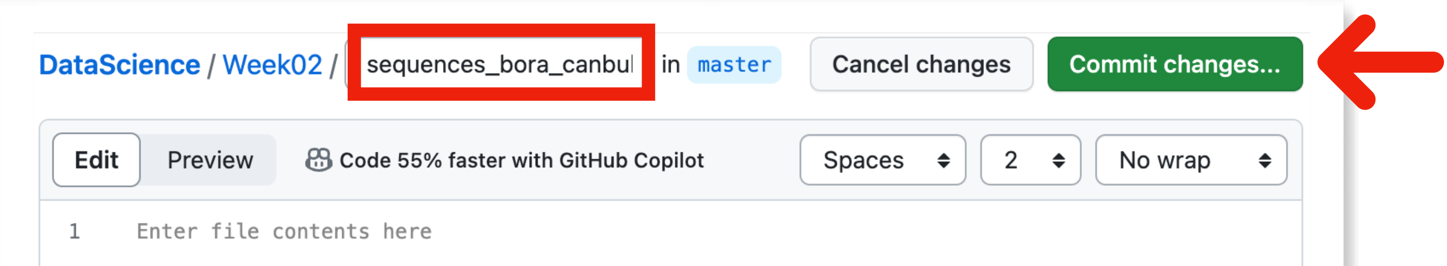
JabbaBC commented 1 minute ago

Describe your changes

Checklist

- ☑ I have read the [CONTRIBUTING]
- ☑ I have performed a self-review of my own code
- ☑ I have run the code locally and it works as expected
- ☑ I have commented my code, particularly in hard-to-understand areas

Screenshots (if appropriate)



sequences_bora_canbul

```
1    Enter file contents here
```

Your Second Homework

- ☑ A list with the name:
  `my_list`
- ☑ A tuple with the name:
  `my_tuple`
- ☑ A set with the name:
  `my_set`
- ☑ A dictionary with the name:
  `my_dict`
- ☑ A function with the name:
  `remove_duplicates (list -> list)`
  to remove duplicate items from a list
- ☑ A function with the name:
  `list_counts (list -> dict)`
  to count the occurrence of each item in a list and return as a dictionary
- ☑ A function with the name:
  `reverse_dict (dict -> dict)`
  to reverse a dictionary, switch values and keys with each other.

Problem Set

1. What is the correct writing of the programming language that we used in this course?

( ) Phyton

( ) Pyhton

( ) Pthyon

( ) Python

2. What is the output of the code below?

```
my_name = "Bora Canbula"
print(my_name[2::-1])
```

( ) alu

( ) ula

( ) roB

( ) Bor

3. Which one is not a valid variable name?

( ) for_

( ) Manisa_Celal_Bayar_University

( ) IF

( ) not

4. What is the output of the code below?

```
for i in range(1, 5):
    print(f"{i:2d}{(i/2):4.2f}", end='')
```

```
( ) 010.50021.00031.50042.00
( )  10.50 21.00 31.50 42.00
( ) 1 0.5 2 1.0 3 1.5 4 2.0
( ) 100.5 201.0 301.5 402.0
```

5. Which one is the correct way to print Bora's age?

```
profs = [
    {"name": "Yener", "age": 25},
    {"name": "Bora", "age": 37},
    {"name": "Ali", "age": 42}
]
```

( ) profs[“Bora”][“age”]

( ) profs[1][1]

( ) profs[1][“age”]

( ) profs.age[name=“Bora”]

6. What is the output of the code below?

```
x = set([int(i/2) for i in range(8)])
print(x)
```

( ) {0, 1, 2, 3, 4, 5, 6, 7}

( ) {0, 1, 2, 3}

( ) {0, 0, 1, 1, 2, 2, 3, 3}

( ) {0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4}

7. What is the output of the code below?

```
x = set(i for i in range(0, 4, 2))
y = set(i for i in range(1, 5, 2))
print(x^y)
```

( ) {0, 1, 2, 3}

( ) {}

( ) {0, 8}

( ) SyntaxError: invalid syntax

8. Which of the following sequences is immutable?

( ) List

( ) Set

( ) Dictionary

( ) String

9. What is the output of the code below?

```
print(int(2_999_999.999))
```

( ) 2

( ) 3000000

( ) ValueError: invalid literal

( ) 2999999

10. What is the output of the code below?

```
x = (1, 5, 1)
print(x, type(x))
```

( ) [1, 2, 3, 4] <class ‘list’>

( ) (1, 5, 1) <class ‘range’>

( ) (1, 5, 1) <class ‘tuple’>

( ) (1, 2, 3, 4) <class ‘set’>

Iterables - Sequences - Iterators

An **iterable** is any object that can be looped over. It represents a collection of elements that can be accessed one by one.

An object is considered iterable if:

- It implements the `__iter__()` method which returns an iterator, or
- It defines the `__getitem()` method that can fetch items using integer indices starting from zero.

A **sequence** is a subtype of iterables. It's an ordered collection of elements that can be indexed by numbers.

- **Ordered:** Elements in a sequence have a specific order.
- **Indexable:** You can get any item using an index `my_sequence[5]`.
- **Slicable:** Supports slicing to get some of items `my_sequence[2:5]`.

An **iterator** is an object that produces items (one at a time) from its associated iterable.

- **Stateful:** An iterator remembers its state between calls. Once an element is consumed, it can't be accessed again without reinitializing the iterator.
- **Lazy Evaluation:** Items are not produced from the source iterable until the iterator's `__next__()` method is called.
- Iterators raise a StopIteration exception when there are no more items to return.
- An iterator's `__iter__()` method returns the iterator object itself.
- While all iterables must be able to produce an iterator (with `__iter__()` method), not all iterators are directly iterable without using a loop.

Numpy Arrays

Numerical Python (**NumPy**)is a powerful library for numerical computing. Its key feature is multi dimensional arrays (**ndarrays**).

Traditional Python Lists

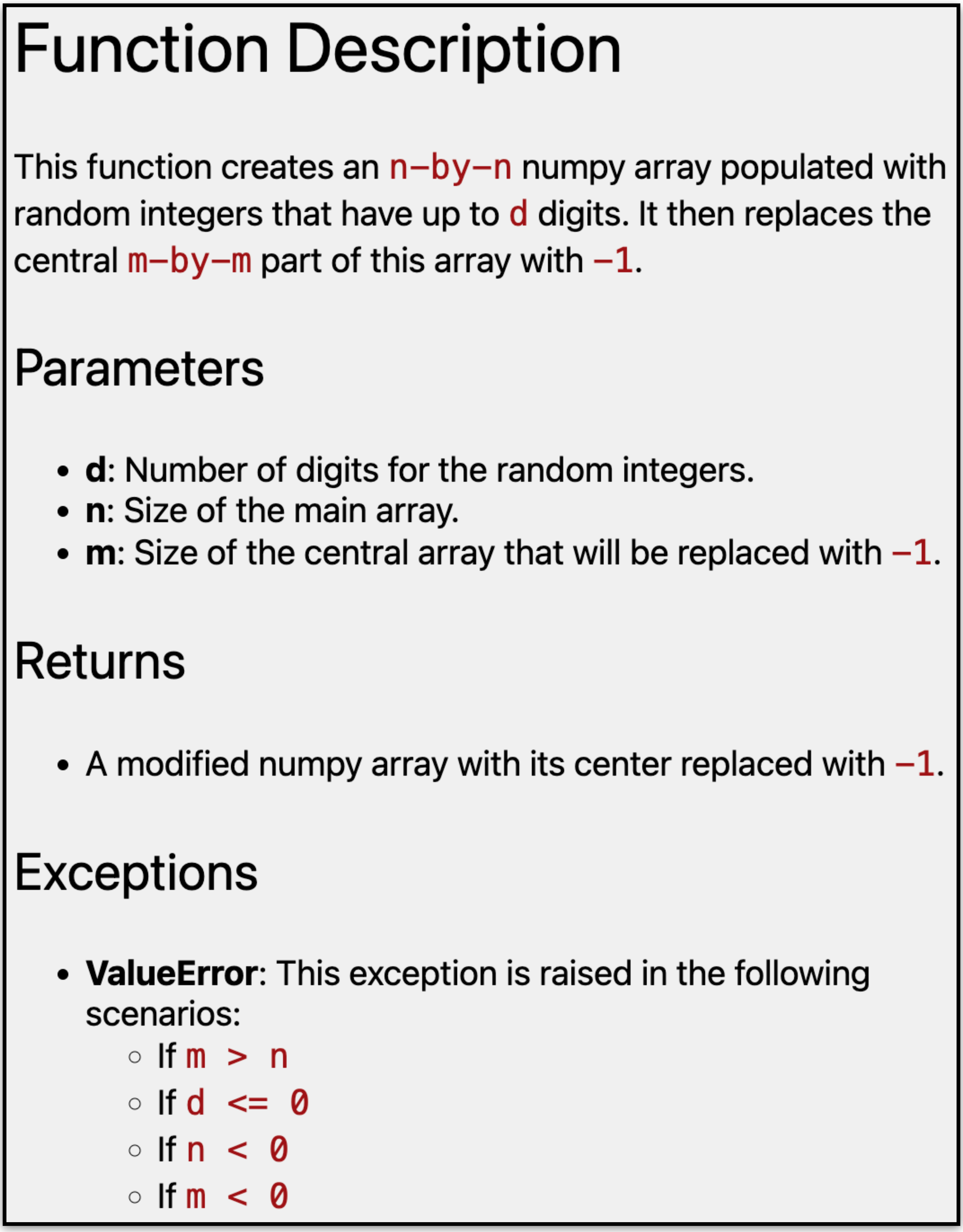
- **Dynamically Typed:** Lists can store elements of mixed types in a single list.
- **Resizable:** Lists can be resized by appending or removing elements.
- **General-purpose:** Lists are general-purpose containers for items of any type.
- **Memory:** Lists have a larger memory overhead because of their general-purpose nature and dynamic typing.
- **Performance:** Basic operations on lists may not be as fast as those on NumPy arrays because they aren't optimized for numerical operations.

NumPy Arrays

- **Typed:** All elements in a NumPy array are of the same type.
- **Size:** The size of a NumPy array is fixed upon creation. However, one can create a new array with a different size, but resizing in-place (like appending in lists) isn't directly supported.
- **Efficiency:** NumPy arrays are memory-efficient as they store elements in contiguous blocks of memory.
- **Performance:** Operations on NumPy arrays are typically faster than lists, especially for numerical tasks, due to optimized C and Fortran extensions.
- **Vectorized Operations:** Supports operations that apply to the entire array without the need for explicit loops (e.g., adding two arrays element-wise).
- **Broadcasting:** Advanced feature allowing operations on arrays of different shapes.
- **Extensive Functionality:** Beyond just array storage, NumPy provides a vast range of mathematical, logical, shape manipulation, and other operations.
- **Interoperability:** Can interface with C, C++, and Fortran code.

Homework

Week03/arrays_firstname_lastname.py



Function Description

This function creates an `n−by−n` numpy array populated with random integers that have up to `d` digits. It then replaces the central `m−by−m` part of this array with -1.

Parameters

- **d**: Number of digits for the random integers.
- **n**: Size of the main array.
- **m**: Size of the central array that will be replaced with -1.

Returns

- A modified numpy array with its center replaced with -1.

Exceptions

- **ValueError**: This exception is raised in the following scenarios:
  - If `m > n`
  - If `d <= 0`
  - If `n < 0`
  - If `m < 0`

Problem Set

1. What is the correct way to create a NumPy array?

( ) np.list([1, 2, 3])

( ) np([1, 2, 3])

( ) np.array([1, 2, 3])

( ) np(array([1, 2, 3]))

2. Which of the following arrays is a 2-D array?

( ) [3, 5]

( ) [[3], [5]]

( ) [{1, 3}, {5, 7}]

( ) [2]

3. What is the correct way to print 5 from the array given below?

a = np.array([[1, 2], [3, 4], [5, 6]])

( ) print(a[3, 1])

( ) print(a[2, 0])

( ) print(a[1, 2])

( ) print(a[1, 3])

4. What is the correct way to print every other item from the array given below?

a = np.arange(5)

( ) print(a[1:3:5])

( ) print(a[::2])

( ) print(a[1:5])

( ) print(a[0:2:4]

5. What does the shape mean of a NumPy array?

( ) Number of columns

( ) Total number of items

( ) Number of items in each dimension

( ) Number of rows

6. What is the output of the code below?

```
n_1 = np.array([1, 2, 3])
n_2 = np.array([4, 5, 6])
n_3 = np.array([7, 8, 9])
print(np.array([n_1, n_2, n_3]).ndim)
```

Your answer:

7. What is the output of the code below?

```
n_1 = np.array([1, 2, 3])
n_2 = np.array([4, 5, 6])
n_3 = np.array([7, 8, 9])
print(np.array([n_1 + n_2 + n_3]).shape)
```

Your answer:

8. Which of the following is created with the code given below?

np.array([[1, 2, 3], [4, 5, 6]])

( ) 1-d array of shape 6 x 1

( ) 2-d array of shape 2 x 3

( ) 3-d array of shape 3 x 2

( ) 3-d array of shape 2 x 3

9. What is the output of the code below?

```
print(np.arange(10).reshape(2, -1))
```

10. What is the output of the code below?

```
Print(np.array([0.5, 1.5, 2.5]).dtype)
```