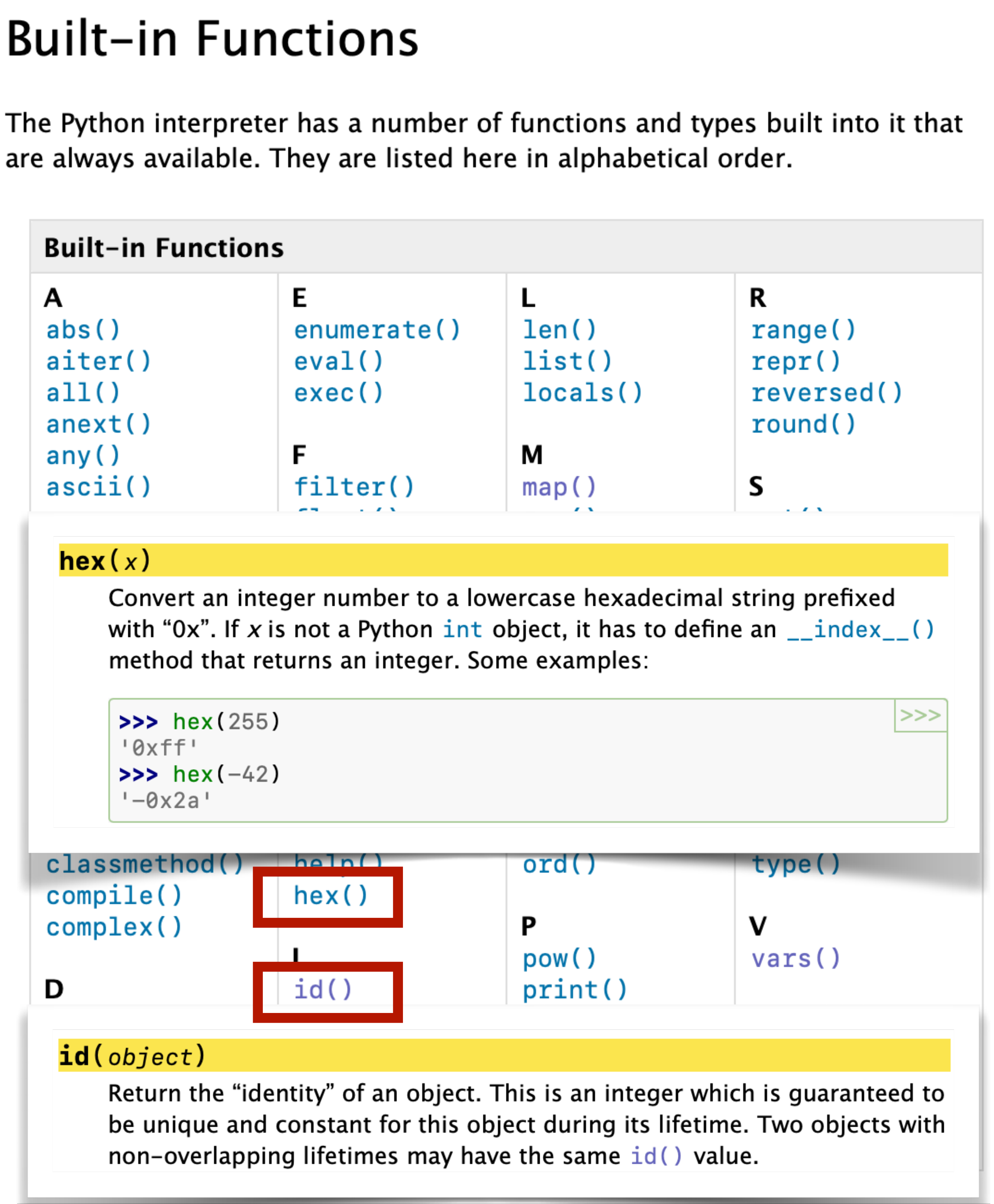
pythonprogramming

Assoc. Prof. Dr. Bora Canbula

github.com/canbula/PythonProgramming

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	**E**	**L**	**R**
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
anext()			round()
any()	**F**	**M**	
ascii()	filter()	map()	**S**
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		**P**	**V**
	I	pow()	vars()
D	id()	print()	

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'
```

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.

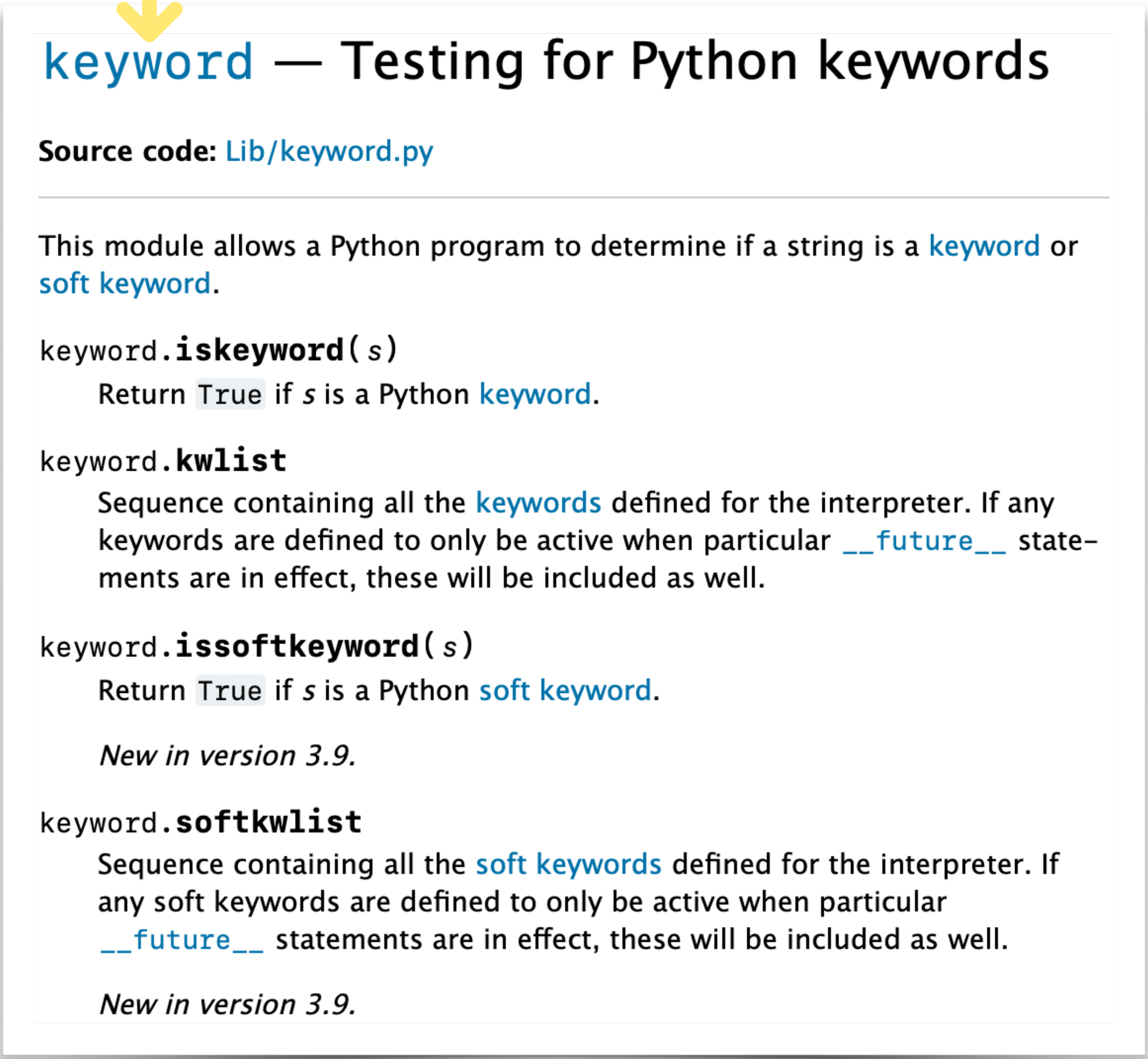
https://docs.python.org/3/library/functions.html

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.

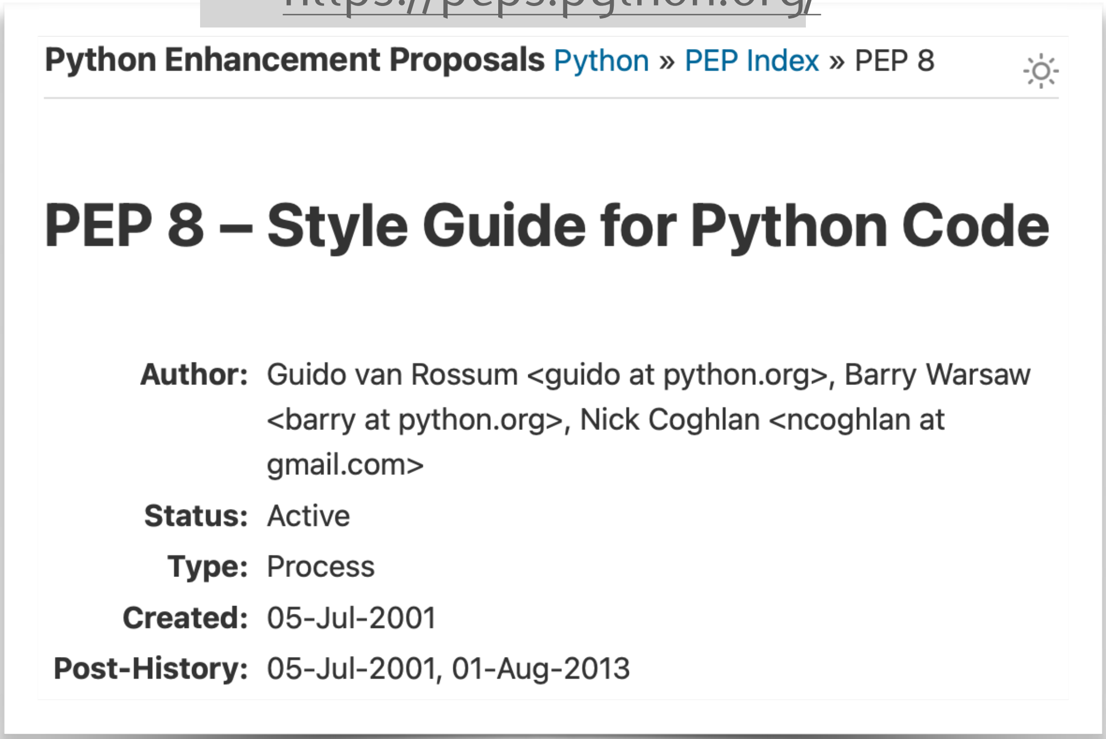
Identifier Names

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.

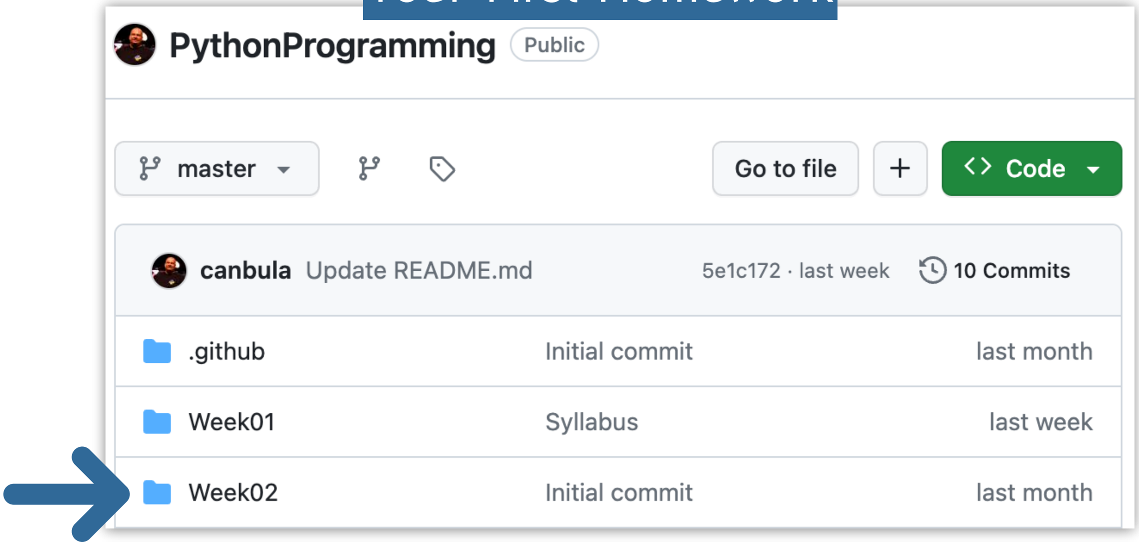
Numeric Types

- Integer
- Float
- Complex
- Boolean

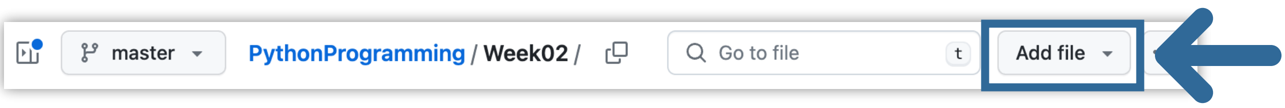
Sequences

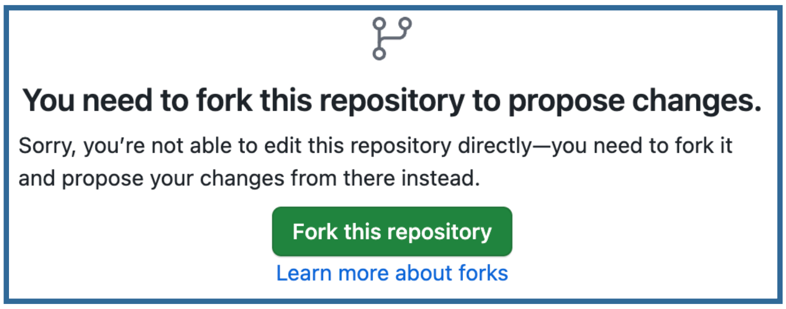
- Strings
- List
- Tuple
- Set
- Dictionary

Your First Homework

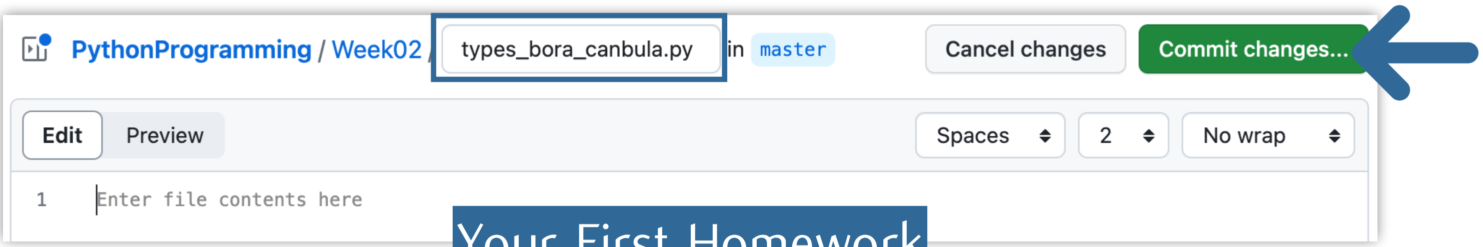


canbula Update README.md	5e1c172 · last week	10 Commits
.github	Initial commit	last month
Week01	Syllabus	last week
Week02	Initial commit	last month




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.


PythonProgramming / Week02 / types_bora_canbula.py in master

```
Enter file contents here
```

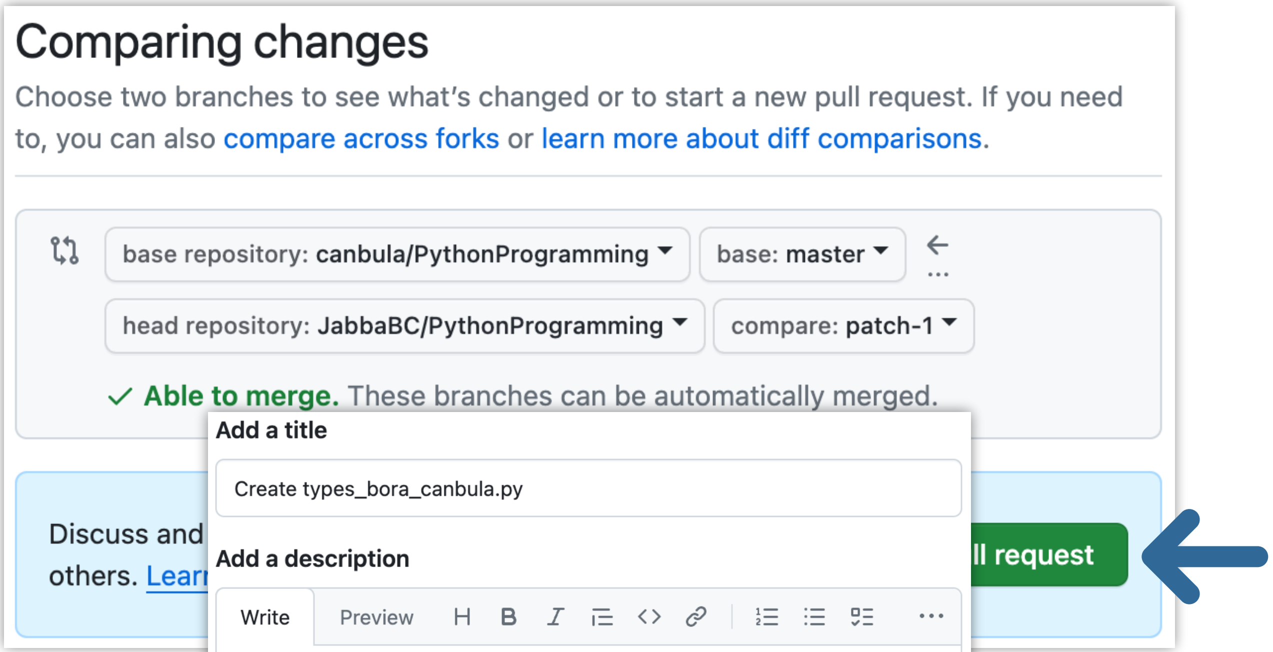
Your First Homework

☑ 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

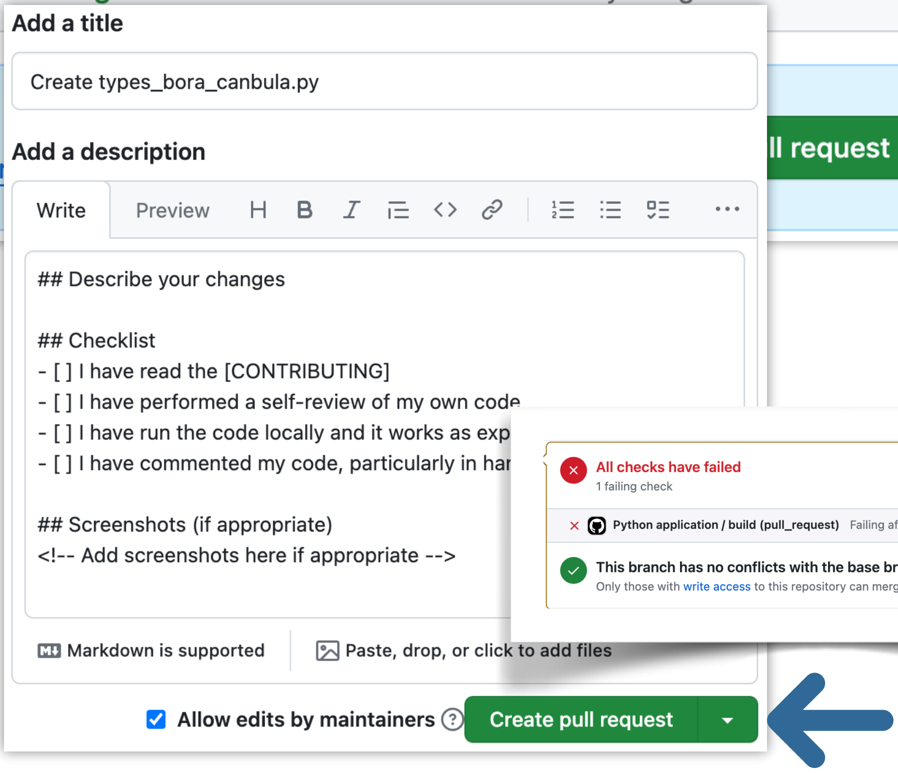
Choose two branches to see what's changed or to start a new pull request. If you need to, you can also compare across forks or learn more about diff comparisons.

base repository: canbula/PythonProgramming   base: master

head repository: JabbaBC/PythonProgramming   compare: patch-1

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

Discuss and … others. Learn…

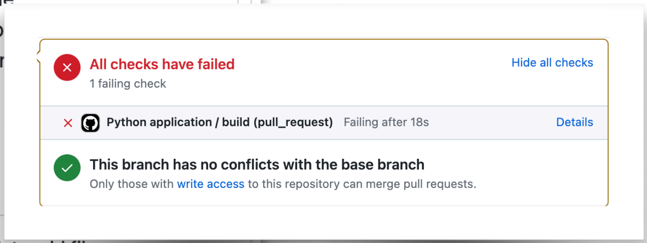

Add a title

Create types_bora_canbula.py

Add a description

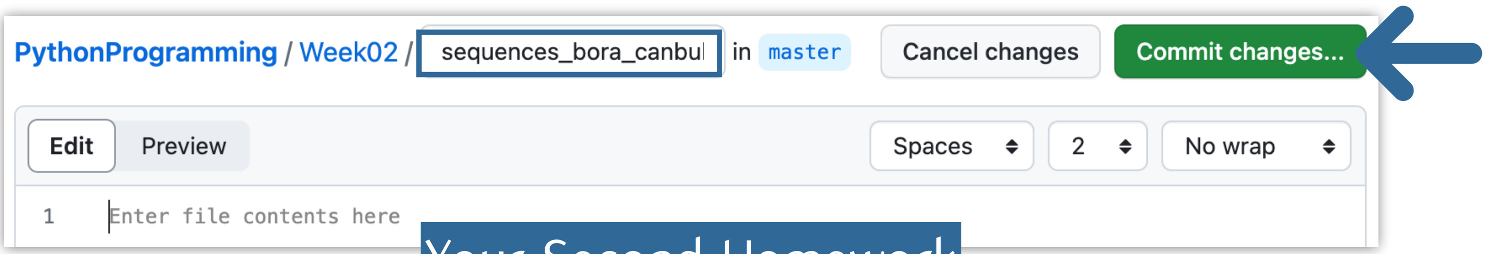
```
## Describe your changes

## 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 exp
- [ ] I have commented my code, particularly in har

## Screenshots (if appropriate)
<!-- Add screenshots here if appropriate -->
```

☑ Allow edits by maintainers


All checks have failed
1 failing check

✕ Python application / build (pull_request) Failing after 18s

✓ This branch has no conflicts with the base branch
Only those with write access to this repository can merge pull requests.


PythonProgramming / Week02 / sequences_bora_canbul in master

```
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'>

Projects

You will pick one software from the list given below. By the end of this semester, you will develop a working clone of your selected software as a desktop or web application. Your team can be up to 5 students.

Notion
Productivity and note-taking application

DB Browser for SQLite
Create, design, and edit database files

Cyberduck
FTP / SFTP client

Engauge Digitizer
Convert graphs and maps into numbers

Zotero
Reference management software

Functions

Functions are defined by using `def` keyword, name and the parenthesized list of formal parameters.

Function names should be lowercase, with words separated by underscores as necessary to improve the readability. PEP 8

Basic Function Definition

```python
def function_name():
    pass
```

Input/Output Arguments

```python
def fn(arg1, arg2):
    return arg1 + arg2
```

Default Values for Arguments

```python
def fn(arg1=0, arg2=0):
    return arg1 + arg2
```

Type Hints and Default Values for Arguments

```python
def fn(arg1: int = 0, arg2: int = 0) -> int:
    return arg1 + arg2
```

PEP 3107

Multiple Type Hints for Arguments

```python
def fn(arg1: int|float, arg2: int|float) -> tuple[float, float]:
    return arg1 + arg2, arg1 * arg2
```

> Python 3.10

Lambda Functions

```python
fn = lambda arg1, arg2: arg1 + arg2
```

Function Docstrings

```python
def fn(arg1=0, arg2=0):
    """This function sums two numbers."""
    return arg1 + arg2
```

PEP 257

Docstrings

PEP 257

A docstring is a string literal that occurs as the first Statement in a module, function, class, or method definition. Such a docstring becomes the `__doc__` special attribute of that object.

One-line Docstrings

```python
def fn(arg1=0, arg2=0):
    """This function sums two numbers."""
    return arg1 + arg2
```

Multi-line Docstrings

```python
def fn(arg1: int = 0, arg2: int = 0) -> int:
    """This function sums two numbers.

    Keyword arguments:
    arg1 -- first number (default 0)
    arg2 -- second number (default 0)
    Return: sum of arg1 and arg2
    """
    return arg1 + arg2
```

Docutils and Sphinx are tools to automatically create documentations

reST (reStructuredText) Format

```python
def fn(arg1: int = 0, arg2: int = 0) -> int:
    """
    This function sums two numbers.

    :param arg1: The first number, def
    :type arg1: int
    :param arg2: The second number, de
    :type arg2: int
    :raises TypeError: Both arguments
    :return: The sum of the two number
    :rtype: int
    """
    if type(arg1) != int or type(arg2)
        raise TypeError("Both argument
    return arg1 + arg2
```

Google Format

```python
"""
This function sums two numbers.

Args:
    arg1 (int): The first number, default is 0
    arg2 (int): The second number, default is 0

Returns:
    int: The sum of the two numbers

Raises:
    TypeError: Both arguments must be integers.
"""
```

Some other formats are Epytext (javadoc), Numpydoc, etc.

Parameter Kinds

PEP 362

Kind describes how argument values are bound to the parameter. The kind can be fixed in the signature of the function.

Standard Binding: Positional-or-Keyword

```python
def fn(arg1=0, arg2=0):
    return arg1 + arg2
```

```python
fn(), fn(3), fn(3, 5), fn(arg1=3)
fn(arg2=5), fn(arg1=3, arg2=5)
```

✓

Positional-or-Keyword & Keyword-Only

```python
def fn(arg1=0, arg2=0, *, arg3=1):
    return (arg1 + arg2) * arg3
```

```python
fn(), fn(3), fn(3, 5), fn(arg1=3), fn(arg2=5)
fn(arg1=3, arg2=5), fn(3, 5, arg3=2)
fn(arg1=3, arg2=5, arg3=2)
fn(arg3=2, arg1=3, arg2=5)
```

✓

```python
fn(3, 5, 2)
fn(arg3=2, 3, 5)
```

✗

Positional-Only & Positional-or-Keyword & Keyword-Only

```python
def fn(arg1=0, arg2=0, /, arg3=1, arg4=1, *, arg5=1, arg6=1):
    return (arg1 + arg2) * arg3 / arg4 * arg5**arg6
```

PEP 457

```python
fn(), fn(3), fn(3, 5), fn(3, 5, 2), fn(3, 5, 2, 4)
fn(3, 5, arg3=2, arg4=4)
fn(3, 5, arg3=2, arg4=4, arg5=7, arg6=8)
fn(3, 5, 2, 4, arg5=7, arg6=8)
```

✓

```python
fn(3, 5, 2, 4, 7, 8)
fn(arg1=3, arg2=5, arg3=2, arg4=4)
fn(arg1=3, arg2=5, arg3=2, arg4=4, arg5=7, arg6=8)
```

✗

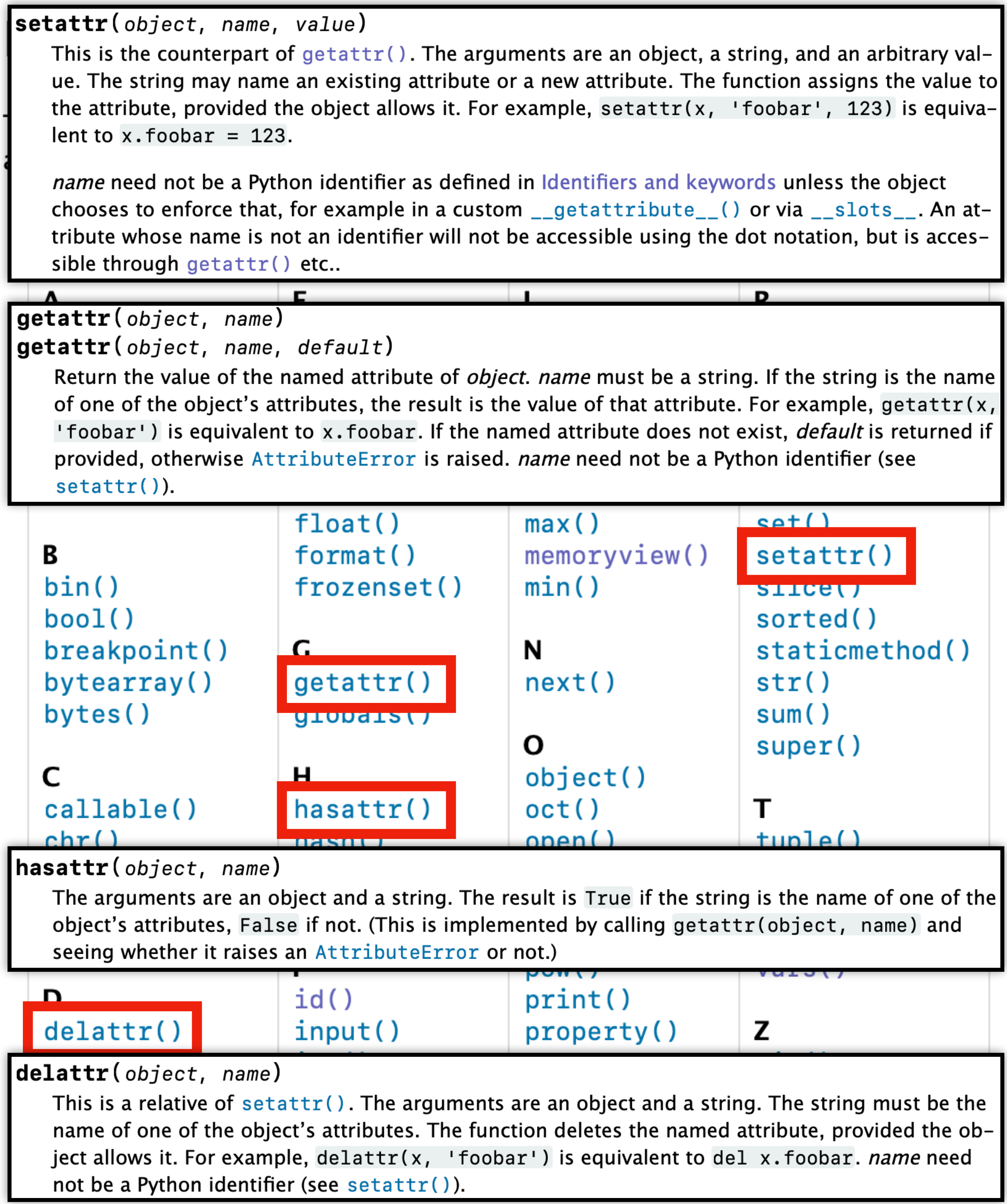
Handle Every Situation

```python
def fn(*args, **kwargs):
    print(args)   # a tuple of positional arguments
    print(kwargs)   # a dictionary of keyword arguments
```

Function Attributes

PEP 232

Functions already have a number of attributes such as `__doc__`, `__annotations__`, `__defaults__`, etc. Like everything in Python, functions are also objects, therefore user can add a dictionary as attributes by using get / set methods to `__dict__`.



setattr(*object, name, value*)

This is the counterpart of `getattr()`. The arguments are an object, a string, and an arbitrary value. The string may name an existing attribute or a new attribute. The function assigns the value to the attribute, provided the object allows it. For example, `setattr(x, 'foobar', 123)` is equivalent to `x.foobar = 123`.

name need not be a Python identifier as defined in Identifiers and keywords unless the object chooses to enforce that, for example in a custom `__getattribute__()` or via `__slots__`. An attribute whose name is not an identifier will not be accessible using the dot notation, but is accessible through `getattr()` etc..

getattr(*object, name*)
getattr(*object, name, default*)

Return the value of the named attribute of *object*. *name* must be a string. If the string is the name of one of the object's attributes, the result is the value of that attribute. For example, `getattr(x, 'foobar')` is equivalent to `x.foobar`. If the named attribute does not exist, *default* is returned if provided, otherwise `AttributeError` is raised. *name* need not be a Python identifier (see `setattr()`).

	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	**G**	**N**	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	**H**	object()	
callable()	hasattr()	oct()	**T**
chr()	hash()	open()	tuple()

hasattr(*object, name*)

The arguments are an object and a string. The result is `True` if the string is the name of one of the object's attributes, `False` if not. (This is implemented by calling `getattr(object, name)` and seeing whether it raises an `AttributeError` or not.)

D	id()	print()	
delattr()	input()	property()	**Z**

delattr(*object, name*)

This is a relative of `setattr()`. The arguments are an object and a string. The string must be the name of one of the object's attributes. The function deletes the named attribute, provided the object allows it. For example, `delattr(x, 'foobar')` is equivalent to `del x.foobar`. *name* need not be a Python identifier (see `setattr()`).

Nested Scopes

PEP 227

Like attributes, function objects can also have methods. These methods can be used as inner functions and can be useful for encapsulation.

```python
def parent_function():
    def nested_function():
        print("I'm a nested function.")
    print("I'm a parent function.")
```

```python
parent_function()
```

✓

```python
parent_function.nested_function()
```

✗

Getter and Setter Methods

```python
def point(x, y):
    def set_x(new_x):
        nonlocal x
        x = new_x
    def set_y(new_y):
        nonlocal y
        y = new_y
    def get():
        return x, y
    point.set_x = set_x
    point.set_y = set_y
    point.get = get
    return point
```

Homework for Functions

Week03/functions_firstname_lastname.py

custom_power

- ✓ A lambda function
- ✓ Two parameters (x and e)
- ✓ x is positional-only
- ✓ e is positional-or-keyword
- ✓ x has the default value 0
- ✓ e has the default value 1
- ✓ Returns x**e

custom_equation

- ✓ A function returns float
- ✓ Five integer parameters (x, y, a, b, c)
- ✓ x is positional-only with default value 0
- ✓ y is positional-only with default value 0
- ✓ a is positional-or-keyword with default value 1
- ✓ b is positional-or-keyword with default value 1
- ✓ c is keyword-only with default value 1
- ✓ Function signature must include all annotations
- ✓ Docstring must be in reST format.
- ✓ Returns (x**a + y**b) / c

fn_w_counter

- ✓ A function returns a tuple of an int and a dictionary
- ✓ Function must count the number of calls with caller information
- ✓ Returning integer is the total number of calls
- ✓ Returning dictionary with string keys and integer values includes the caller (_ _name_ _) as key, the number of call coming from this caller as value.

Examples

```
custom_power(2) == 2
custom_power(2, 3) == 8
custom_power(2, e=2) == 4
custom_equation(2, 3) == 5.0
custom_equation(2, 3, 2) == 7.0
custom_equation(2, 3, 2, 3) == 31.0
custom_equation(3, 5, a=2, b=3, c=4) == 33.5
custom_equation(3, 5, 2, b=3, c=4) == 33.5
custom_equation(3, 5, 2, 3, c=4) == 33.5
for i in range(10):
    fn_w_counter()
fn_w_counter() == (11, {'__main__': 11})
```