Numerical Analysis

Assoc. Prof. Dr. Bora Canbula

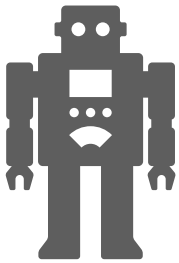
https://github.com/canbula/NumericalAnalysis/

Python Basics
Key Features of NumPy
Binary Representation of Numbers
IEEE 754 Representation of Numbers
Precisions in IEEE 754 Representation
Introduction to Numerical Derivatives
Finite Difference Approach
System of Linear Equations
Bisection Method
Newton - Raphson Method
Introduction to Numerical Integration
Gaussian Quadrature Method
System of Nonlinear Equations
Review and Applications of Topics

Python Basics

Natural Languages vs. Programming Languages

A language is a tool for expressing and recording thoughts.



Computers have their own language called **machine** language. Machine languages are created by humans, no computer is currently capable of creating a new language. A complete set of known commands is called an instruction list (IL).

The difference is that human languages developed naturally. They are still evolving, new words are created every day as old words disappear. These languages are called **natural** languages.

Elements of a Language

- **Alphabet** is a set of symbols to build words of a certain language.
- **Lexis** is a set of words the language offers its users.
- **Syntax** is a set of rules used to determine if a certain string of words forms a valid sentence.
- **Semantics** is a set of rules determining if a certain phrase makes sense.

Machine Language vs. High-Level Language

The IL is the alphabet of a machine language. It's the computer's mother tongue.

High-level programming language enables humans to write their programs and computers to execute the programs. It is much more complex than those offered by ILs.

A program written in a high-level programming language is called a **source code**. Similarly, the file containing the source code is called the **source file**.

Python Basics

Compilation vs. Interpretation

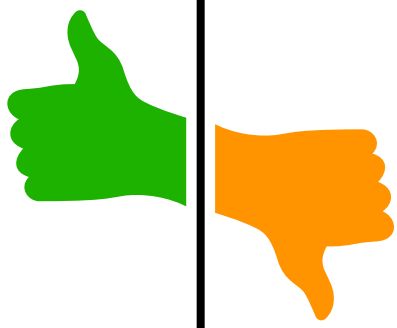
There are two different ways of transforming a program from a high-level programming language into machine language:

Compilation: The source code is translated once by getting a file containing the machine code.

Interpretation: The source code is interpreted every time it is intended to be executed.

Compilation

- The execution of the translated code is usually faster.
- Only the user has to have the compiler. The end user may use the code without it.
- The translated code is stored using machine language. Your code are likely to remain your secret.



- The compilation itself may be a very time-consuming process
- You have to have as many compilers as hardware platforms you want your code to be run on.

Interpretation

- You can run the code as soon as you complete it, there are no additional phases of translation.
- The code is stored using programming language, not machine language. You don't compile your code for each different architecture.

- Don't expect interpretation to ramp up your code to high speed
- Both you and the end user have the interpreter to run your code.

Python Basics

What is Python?

Python is a widely-used, interpreted, object-oriented, and high-level programming language with dynamic semantics, used for general-purpose programming.

Python was created by Guido van Rossum. The name of the Python programming language comes from an old BBC television comedy sketch series called Monty Python's Flying Circus.

Python Goals

- an **easy and intuitive** language just as powerful as those of the major competitors
- **open source**, so anyone can contribute to its development
- code that is as **understandable** as plain English
- **suitable for everyday tasks**, allowing for short development times

Why Python?

- easy to learn
- easy to teach
- easy to use
- easy to understand
- easy to obtain, install and deploy

Why not Python?

- low-level programming
- applications for mobile devices

Python Basics

Python Implementations

An implementation refers to a program or environment, which provides support for the execution of programs written in the Python language.

- **CPython** is the traditional implementation of Python and it's most often called just "Python".
- **Cython** is a solution which translate Python code into "C" to make it run much faster than pure Python.
- **Jython** is an implementation follows only Python 2, not Python 3, written in Java.
- **PyPy** represents a Python environment written in Python-like language named RPython (Restricted Python), which is actually a subset of Python.
- **MicroPython** is an implementation of Python 3 that is optimized to run on microcontrollers.

Start Coding with Python

- **Editor** will support you in writing the code. The Python 3 standard installation contains a very simple application named IDLE (Integrated Development and Learning Environment).
- **Console** is a terminal in which you can launch your code.
- **Debugger** is a tool, which launches your code step-by-step to allow you to inspect it.

first.py

```
print("Python is the best!")
```

Python Basics

Function Name
A function can cause some effect
or evaluate a value, or both.

Where do functions come from?
- From Python itself
- From modules
- From your code

first.py

```
print("Python is the best!")
```

Argument
- Positional arguments
- Keyword arguments

`print``(*objects, sep=' ', end='\n', file=None, flush=False)`

Print *objects* to the text stream *file*, separated by *sep* and followed by *end*. *sep*, *end*, *file*, and *flush*, if present, must be given as keyword arguments.

All non-keyword arguments are converted to strings like `str()` does and written to the stream, separated by *sep* and followed by *end*. Both *sep* and *end* must be strings; they can also be `None`, which means to use the default values. If no *objects* are given, `print()` will just write *end*.

The *file* argument must be an object with a `write(string)` method; if it is not present or `None`, `sys.stdout` will be used. Since printed arguments are converted to text strings, `print()` cannot be used with binary mode file objects. For these, use `file.write(...)` instead.

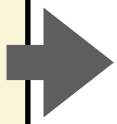
Output buffering is usually determined by *file*. However, if *flush* is true, the stream is forcibly flushed.

Literals

A literal is data whose values are determined by the literal itself. Literals are used to encode data and put them into code.

literals.py

```
print("7")
print(7)
print(7.0)
print(7j)
print(True)
print(0b10)
print(0o10)
print(0x10)
print(7.4e3)
```



- String
- Integer
- Float
- Complex
- Boolean
- Binary
- Octal
- Hexadecimal
- Scientific Notation

Python Basics

Basic Operators

An operator is a symbol of the programming language, which is able to operate on the values.

Multiplication

```
print(2 * 3)      Integer
print(2 * 3.0)    Float
print(2.0 * 3)    Float
print(2.0 * 3.0)  Float
```

Division

```
print(6 / 3)      Float
print(6 / 3.0)    Float
print(6.0 / 3)    Float
print(6.0 / 3.0)  Float
```

Exponentiation

```
print(2**3)       Integer
print(2**3.0)     Float
print(2.0**3)     Float
print(2.0**3.0)   Float
```

Floor Division

```
print(6 // 3)     Integer
print(6 // 3.0)   Float
print(6.0 // 3)   Float
print(6.0 // 3.0) Float
```

Modulo

```
print(6 % 3)      Integer
print(6 % 3.0)    Float
print(6.0 % 3)    Float
print(6.0 % 3.0)  Float
```

Addition

```
print(-8 + 4)     Integer
print(-4.0 + 8)   Float
```

Operator Priorities

An operator is a symbol of the programming language, which is able to operate on the values.

priorities.py

```
print(9 % 6 % 2)
print(2**2**3)
print(2 * 3 % 5)
print(-3 * 2)
print(-2 * 3)
print(-(2 * 3))
```

- ** (right-sided binding)
- + (unary)
- - (unary)
- *
- /
- //
- % (left-sided binding)
- + (binary)
- - (binary)

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

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()	help()	ord()	type()
compile()	hex()		
complex()		P	V
	I	pow()	vars()
D	id()	print()	

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.

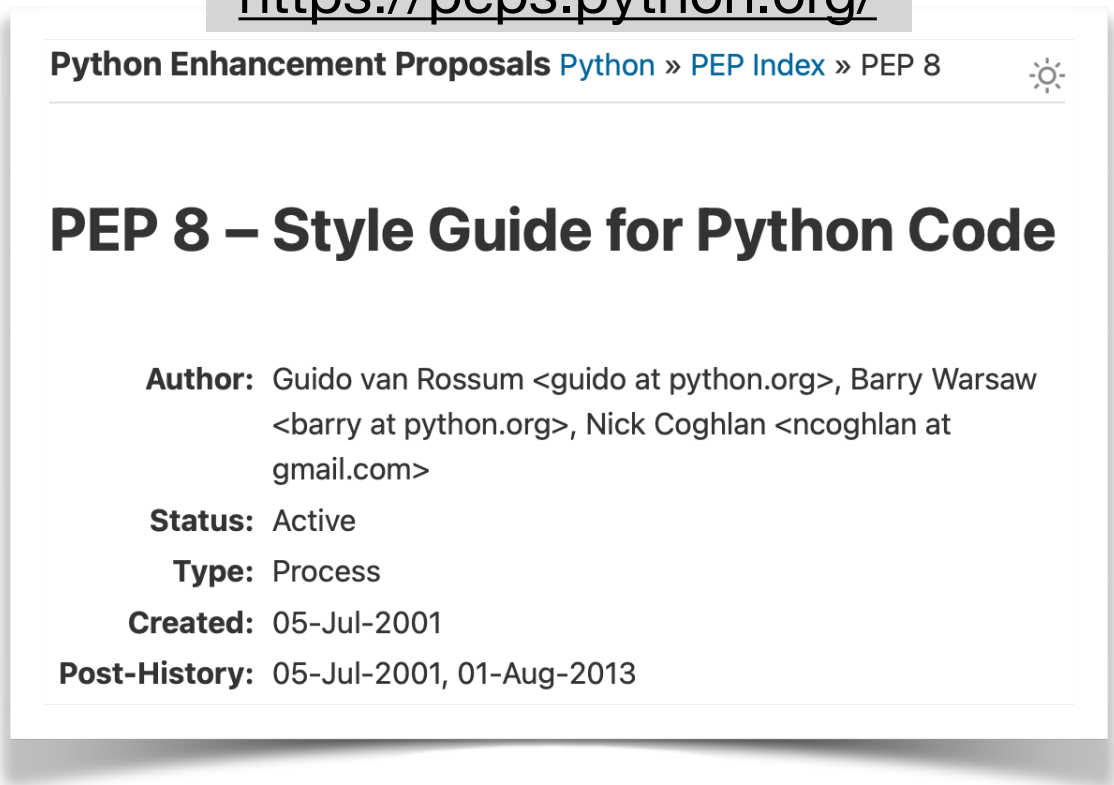
Identifier Names

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

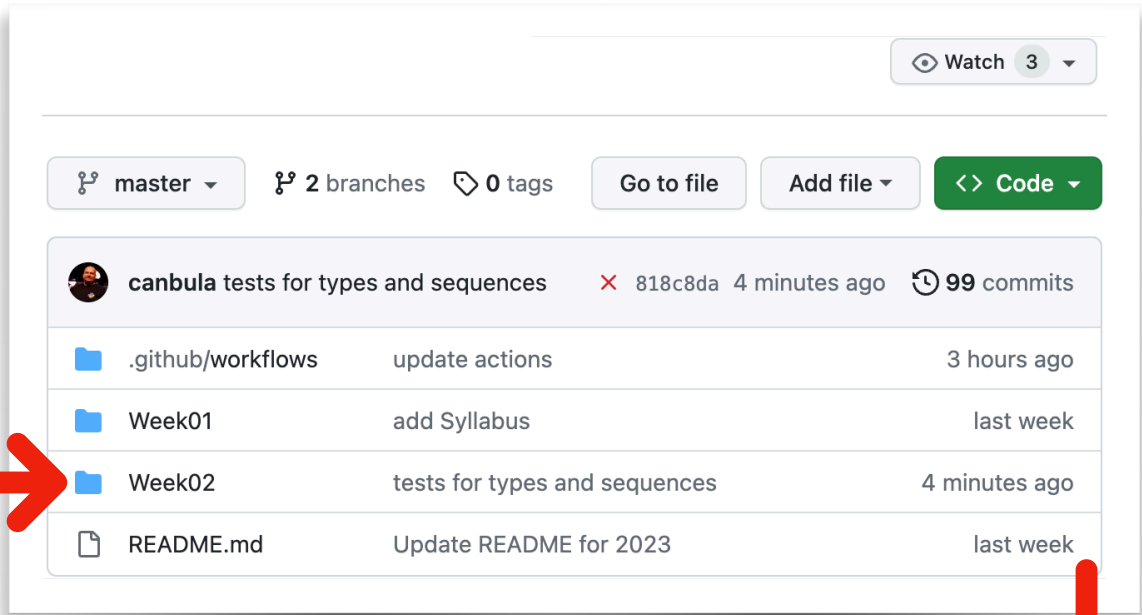
- Names to Avoid
  Never use the characters ‘l’ (lowercase letter el), ‘O’ (uppercase letter oh), or ‘I’ (uppercase letter eye) as single character variable names.
- Packages
  Short, all-lowercase names without underscores
- Modules
  Short, all-lowercase names, can have underscores
- Classes
  CapWords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

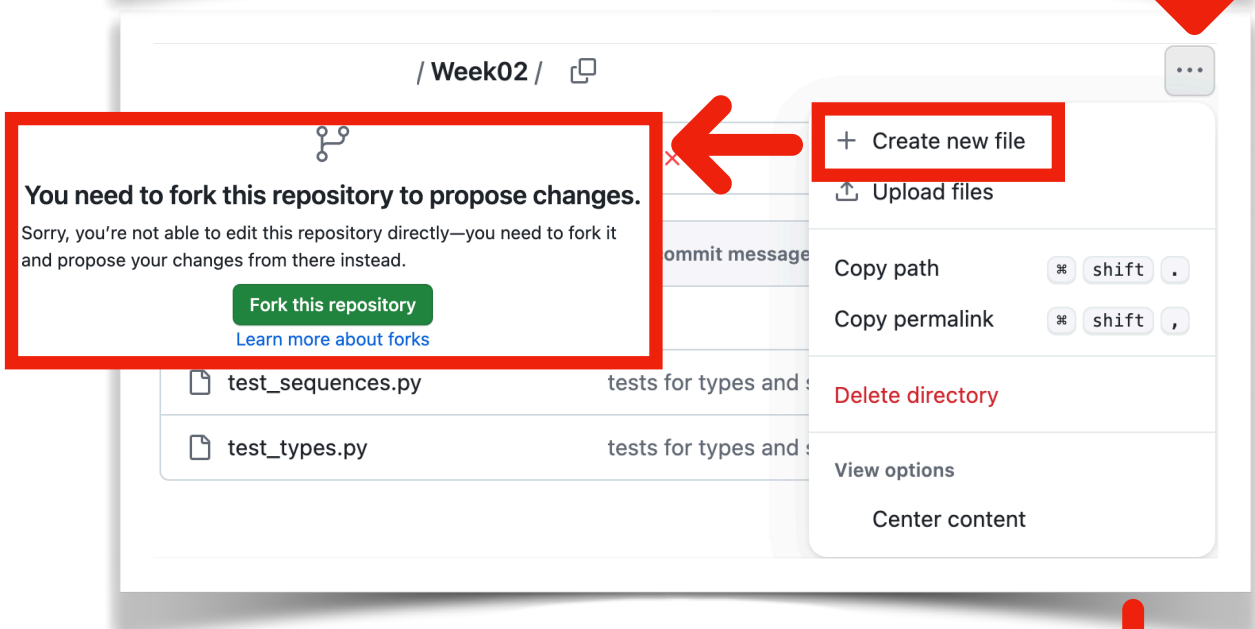
Leading and Trailing Underscores

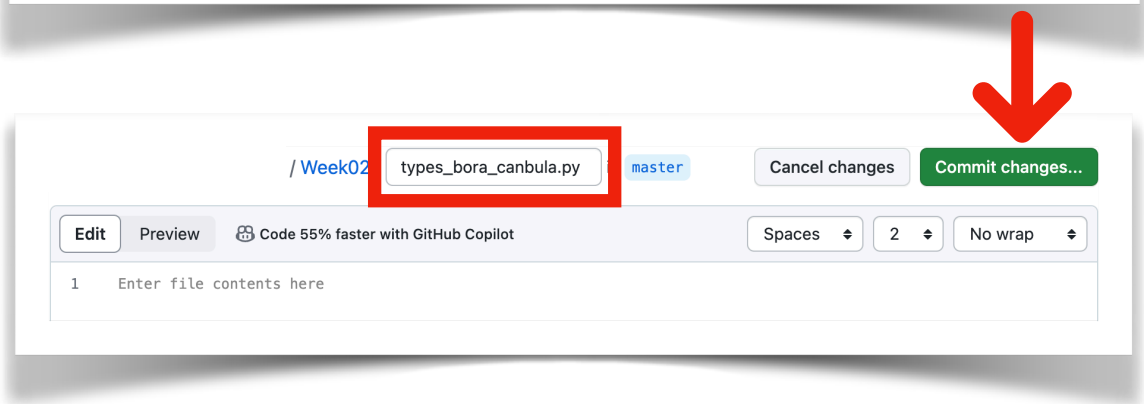
- `_single_leading_underscore`
  Weak “internal use” indicator.
  `from M import *` does not import objects whose names start with an underscore.
- `single_trailing_underscore_`
  Used by convention to avoid conflicts with keyword.
- `__double_leading_underscore`
  When naming a class attribute, invokes 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.

Your First Homework

Week02/types_first_last.py







- 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

Sequences

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
```

Sequences

```
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

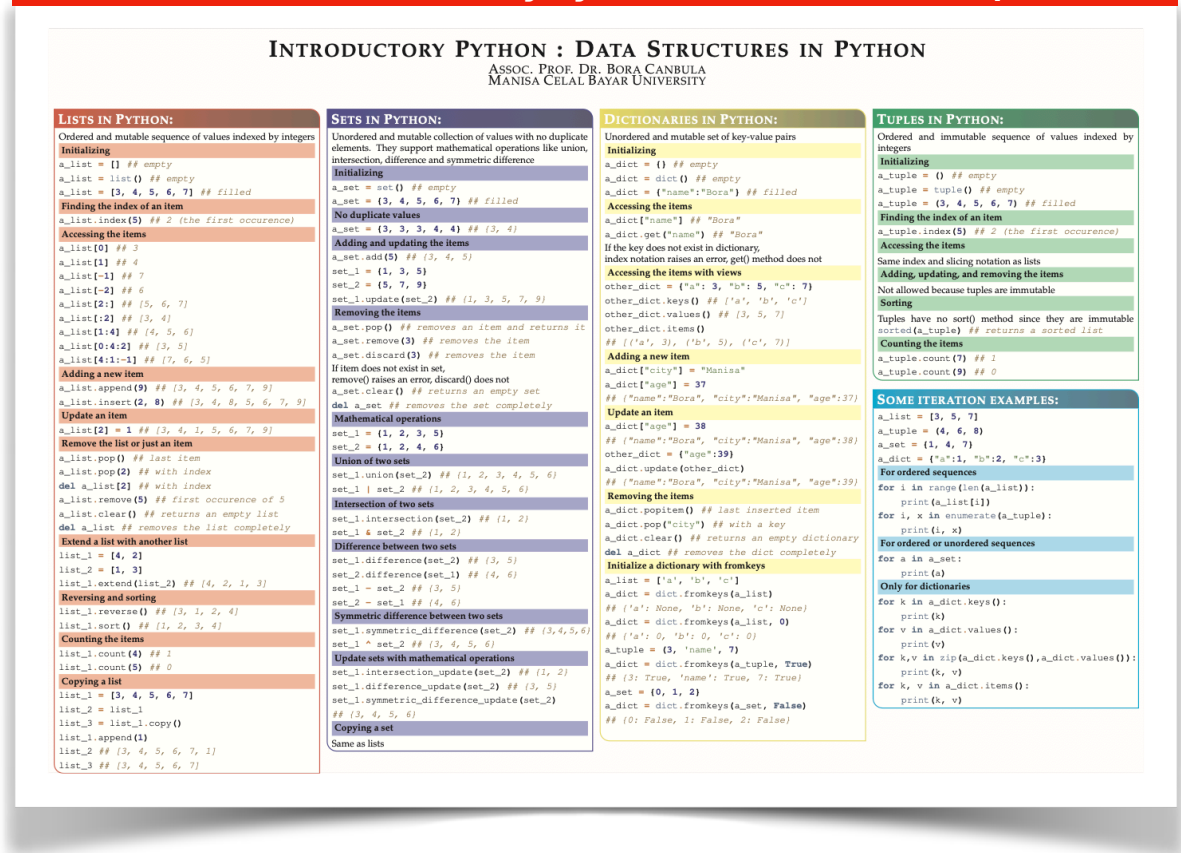
```
list_1.count(4) ## 1
list_1.count(5) ## 0
```

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]
```

Sequences

Week03/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)
```

Week03/sequences_first_last.py

```python
def remove_duplicates(seq: list) -> list:
    """
    This function removes duplicates from a list.
    """
    return ...

def list_counts(seq: list) -> dict:
    """
    This function counts the number of
    occurrences of each item in a list.
    """
    return ...

def reverse_dict(d: dict) -> dict:
    """
    This function reverses the keys
    and values of a dictionary.
    """
    return ...
```

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

Submit your work to GitHub

Week04/arrays_firstname_lastname.py

Function Description

replace_center_with_minus_one(d, n, m)

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

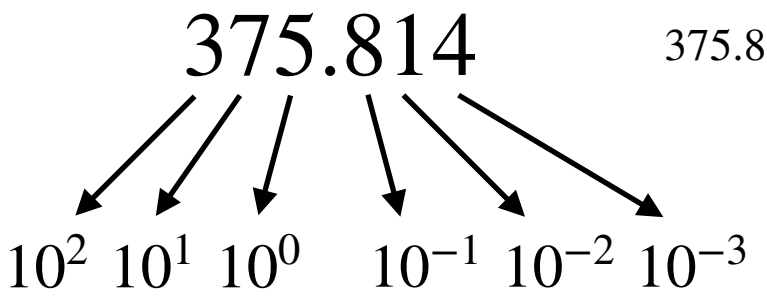
9. What is the output of the code below?

```
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)
```

Binary Representation of Floating-Point Numbers



$$375.814 = (3 \cdot 10^2) + (7 \cdot 10^1) + (5 \cdot 10^0) + (8 \cdot 10^{-1}) + (1 \cdot 10^{-2}) + (4 \cdot 10^{-3})$$

Integer Part

$$\begin{aligned} 375 \,\%\, 2 &= 1 \\ 187 \,\%\, 2 &= 1 \\ 93 \,\%\, 2 &= 1 \\ 46 \,\%\, 2 &= 0 \\ 23 \,\%\, 2 &= 1 \\ 11 \,\%\, 2 &= 1 \\ 5 \,\%\, 2 &= 1 \\ 2 \,\%\, 2 &= 0 \\ 1 & \end{aligned}$$

Decimal Part

$$\begin{aligned} 0.814 \cdot 2 &= 1.628 \rightarrow 1 \\ 0.628 \cdot 2 &= 1.256 \rightarrow 1 \\ 0.256 \cdot 2 &= 0.512 \rightarrow 0 \\ 0.512 \cdot 2 &= 1.024 \rightarrow 1 \\ 0.024 \cdot 2 &= 0.048 \rightarrow 0 \\ 0.048 \cdot 2 &= 0.096 \rightarrow 0 \\ 0.096 \cdot 2 &= 0.192 \rightarrow 0 \\ 0.192 \cdot 2 &= 0.384 \rightarrow 0 \\ 0.384 \cdot 2 &= 0.768 \rightarrow 0 \\ 0.768 \cdot 2 &= 1.536 \rightarrow 1 \\ 0.036 \cdot 2 &= 0.072 \rightarrow 0 \\ &\vdots \end{aligned}$$

$$(375.814)_{10} = (101110111.11010000010\ldots)_2$$

Error

Homework

Week05/bra_firstname_lastname.py

Submit your work to GitHub

Binary Representation API

Your task is to create a Python application that exposes an API endpoint to convert floating-point numbers into their binary representation. This application will be a Flask web service that can accept GET requests with a floating-point number and respond with its binary representation. The binary representation should separate the integer and fractional parts with a dot.

Objectives

- Implement a BinaryRepresentation class that can take a float and provide methods to convert both the integer and decimal parts into binary.
- Develop a Flask API with a route that accepts a floating-point number as a query parameter and returns its binary representation.
- Ensure that your code passes a series of tests that will be run against it.
- All the details that you can't understand from this assignment should be extracted from the tests: **Week05/test_bra.py**

Requirements

1. **BinaryRepresentation Class**
   - The class should be initialized with one float parameter.
   - Implement a method **integer2binary** that converts the integer part of the float to binary.
   - Implement a method **decimal2binary** that converts the decimal part of the float to binary up to 10 places.
   - Implement the **__str__** method to return the binary representation as a string, separating the integer and decimal parts with a dot.
2. **Flask API**
   - Set up a Flask application.
   - Define a route that listens to **GET** requests and expects a number query parameter.
   - Validate that the number parameter is a **float** and return an appropriate response if not.
   - Utilize the **BinaryRepresentation** class to return the binary representation of the number.
3. **Error Handling**
   - Ensure that the application correctly handles cases where the number is not provided, or is not a valid float.

Problem Set

1. In binary system, which of the following digits are used to represent a number?

( ) 1 and 2
( ) 0 and 1
( ) 0, 1 and 2
( ) A and B

2. Which of the following codes gives a binary representation of 97?

( ) binary(97)
( ) (97).binary()
( ) f"{97:b}"
( ) to_binary(97)

3. What is the name of the NumPy method which converts a number to binary system?

( ) np.binary()
( ) np.bin()
( ) np.binary_representation()
( ) np.binary_repr()

4. The code given below produces this output:

```
> 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 = 97
```

Complete the code with appropriate statements for the lines given with (1) and (2).

```
n = 16;r = 97;r_0 = r;b = [0]*n
for i in range(n-1, -1, -1):
    x = 2**i
    if r >= x:
        (1)
        (2)
b = b[::-1]
print(*b, end='')
print(f" = {r_0}")
```

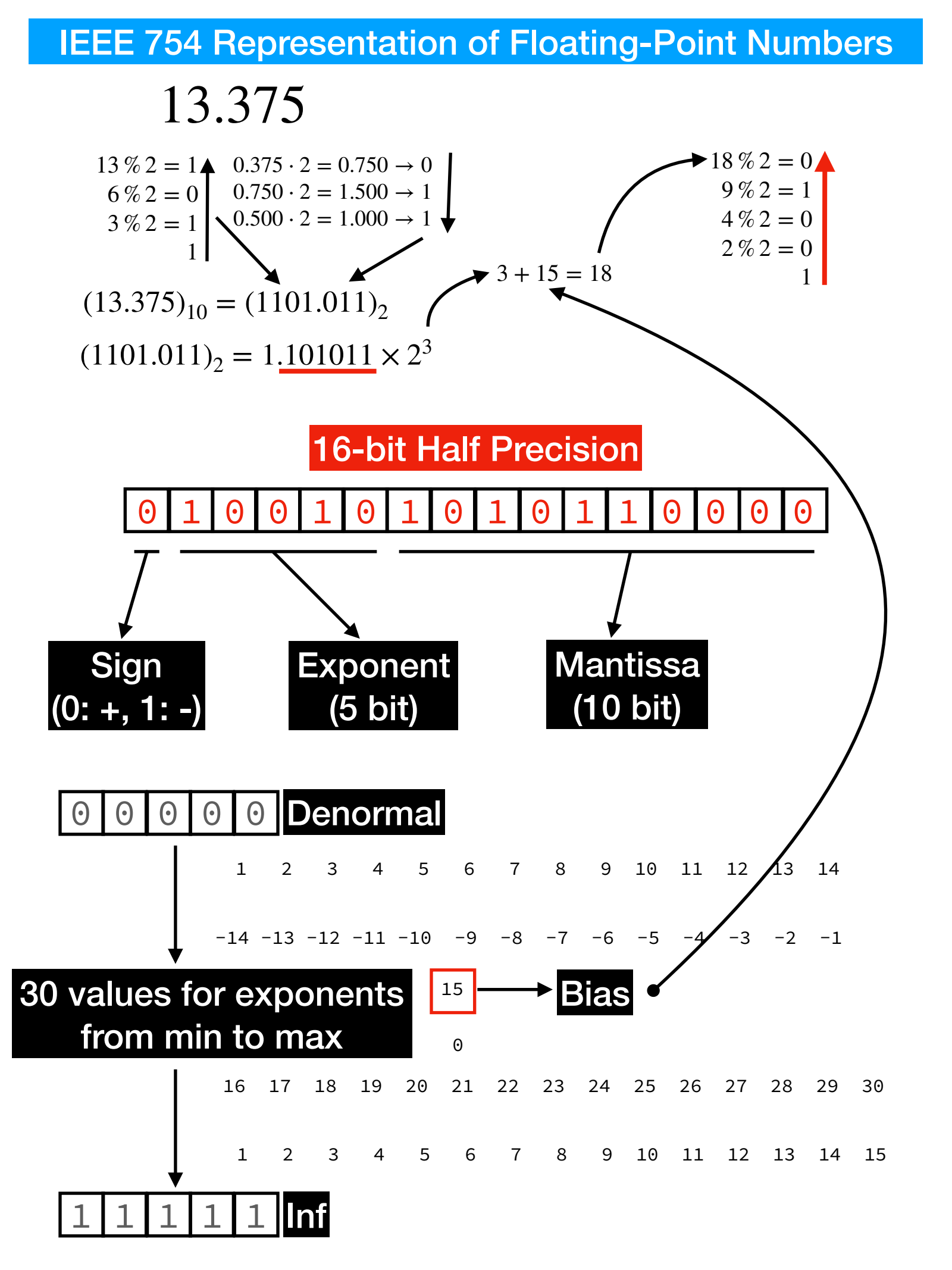
5. Modify the code given in question **4** to avoid fixing the number of digits (**n**). Hint: use bit_length() method of integer object.

6. Use the codes given in the question **4** as a starting point and write Python codes which converts the decimal of a base-10 number into binary system.

7. Try to write a general function which converts a base-10 floating point number into any base including the decimal part.

```
def to_any_base(r: float, b: int) -> str:
    '''This function returns the base-b '''
    '''conversion of r, which is a      '''
    '''floating-point number.           '''
    '''Example:                         '''
    '''  to_any_base(3.5, 2) -> '11.1'  '''
```

IEEE 754 Representation of Floating-Point Numbers



Homework

Week06/halfprecision_firstname_lastname.py

Submit your work to GitHub

Implementing a Half-Precision Floating Point Converter

In computing, half-precision floating-point is a binary floating-point computer number format that occupies 16 bits (two bytes in modern computers) in computer memory. They are designed for use in situations where a wide dynamic range is not required, and where storage space is at a premium.

Your task is to implement a Python class named **HalfPrecision** that converts a Python floating-point number (**float**) to its binary **string** representation in half-precision floating-point format according to IEEE 754 standards.

All the details that you can't understand from this assignment should be extracted from the tests: **Week06/test_halfprecision.py**

Requirements

- Implement the class **HalfPrecision** with an **__init__** method and a **__str__** method.
- The **__init__** method should take a single argument: the number to be converted and also raises a **TypeError** if the number is **not a float**.
- The __str__ method should return a 16-character string representing the binary format as:
    - The first bit is the sign bit.
    - The following five bits represent the exponent.
    - The last ten bits represent the mantissa.
- Your class should correctly handle special numbers such as NaN (Not a Number), infinities, and denormal numbers. Only infinities will be tested at this stage, other special numbers will be studied next week in the classroom.

Problem Set

1. Find the smallest and the largest value that you can represent with 16-bit IEEE 754 standard?

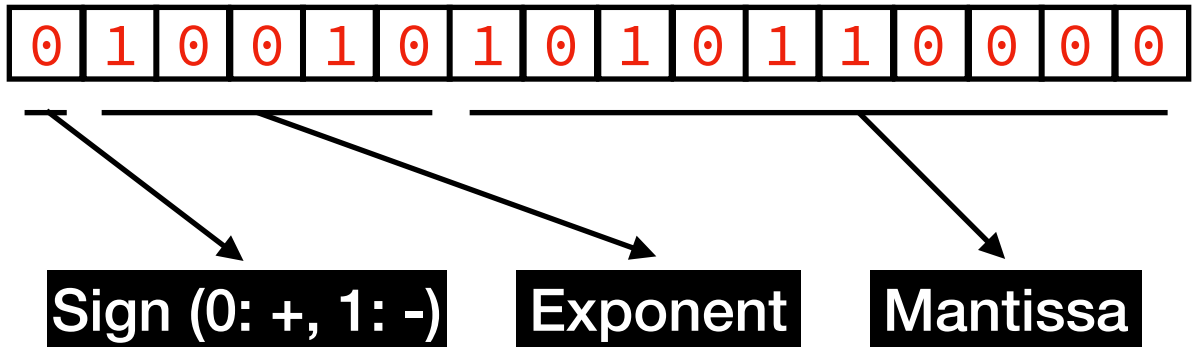
2. Find the 16 bit IEEE 754 representation of -5.875.

3. Calculate the error if we use 16 bit IEEE 754 representation to store the value 0.17 in memory.

4. Use a custom IEEE 754 representation as 1-bit for the sign of the number and (4-bit exponent) + (20-bit mantissa). Convert 0.17 into this representation and compare the result with the previous question.

5. Calculate the bias for the 8 bit exponent part.

Precisions for IEEE 754 Representation



IEEE 754 Precisions

Precision	Sign	Exponent	Mantissa	Total
Half	1	5	10	16
Single	1	8	23	32
Double	1	11	52	64
Quadruple	1	15	112	128
Octuple	1	19	236	256

Alternative method to convert floats to binary?

13.375

$13.375 \cdot 2 = 26.75$
$26.750 \cdot 2 = 53.50$
$53.500 \cdot 2 = 107.0$

Number times 2
To get an integer

$(13.375)_{10} \cdot 2^3 = 107$

$107 \,\%\, 2 = 1$
$53 \,\%\, 2 = 1$
$26 \,\%\, 2 = 0$
$13 \,\%\, 2 = 1$
$6 \,\%\, 2 = 0$
$3 \,\%\, 2 = 1$
1

13.375 **Legacy Way**

$13 \,\%\, 2 = 1$
$6 \,\%\, 2 = 0$
$3 \,\%\, 2 = 1$
1

$0.375 \cdot 2 = 0.750 \rightarrow 0$
$0.750 \cdot 2 = 1.500 \rightarrow 1$
$0.500 \cdot 2 = 1.000 \rightarrow 1$

$(13.375)_{10} = (1101.011)_2$

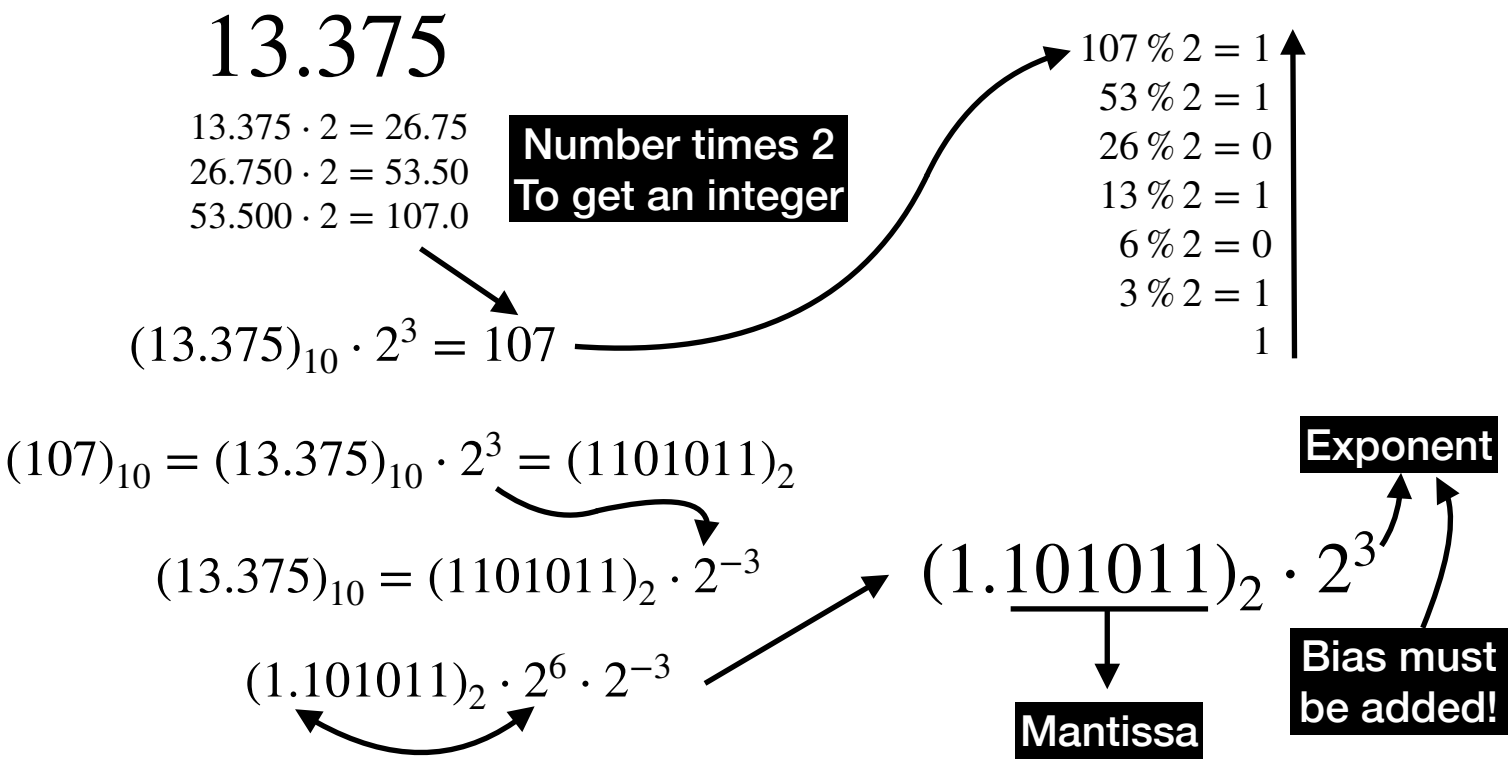
$(107)_{10} = (13.375)_{10} \cdot 2^3 = (1101011)_2$

$(13.375)_{10} = (1101011)_2 \cdot 2^{-3}$

$(13.375)_{10} = (1101.011)_2$

The power must be added to exponent

Precisions for IEEE 754 Representation



- ✓ Define the precisions
- ✓ Enable the custom precision
- ✓ Calculate the bias
- ✓ Find the power of 2 to scale up the number to an integer
- ✓ Use the bias and scale to calculate the exponent
- ✓ Convert exponent to binary
- ✓ Fill the mantissa with trailing zeros
- ✓ Find the sign of the number
- ✓ Return the result as a string with __str__ method

- ✓ Input validation
- ✓ Edge cases such as Inf, NaN, and signed zero
- ✓ Find the normalization range, raise error for denormals

Convert the IEEE 754 Representation to Float?

Problem Set

1. The numbers used in the following equation are given in Half Precision IEEE 754 format, but in hexadecimal notation. Please find the result as a base-10 number.

67C8 + 3C00 =

2. Suppose that we want to save the value 0.1 in our PC. How many bits do we need for the mantissa part?

3. Which one is the correct representation of zero in IEEE 754 half precision?

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

or

0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0

Explain your answer.

4. You can find the current version of the file ieee754.py in the folder Week06 of GitHub repo of this course. List the weak points of this code that must be fixed.

5. Using NumPy arrays to save the zeros and ones in ieee754.py, was it a correct choice or not? Explain your answer.