Python for everyone

Basics

- `print()` and `input()`
- immutable types: int, float, bool, string, tuple
- mutable types: list, dictionary
- variables
- comments
- arithmetic operators: `+, -, *, /, //, %, **`
- comparison operators: `==, !=, <, >, <=, >=`
- boolean operators: `not, and, or`
- sequence operators: `+, *, in, not in` (strings, tuples, lists)
- standard library modules: `math, random, time`
- `if-elif-else`
- `while`
- `for`
- indented block of code

Functions

- user-defined vs. built-in
- def
- parameters:
  - positional
  - keyword
- the return statement
- local and global variables

Accessing documentation

```
x?                                  # provides a short
documentation of x
dir(str)                            # list of methods that
of a class/object, str in this case
```

Sequences

- strings and tuples (imutable) and lists (mutable)
- function len
- operators: `+, *,  in, not in`
- indexing with `[]` (starts at index **0**, the last element is at index **-1**)
- slicing `[:]`
- iterating over elements (for loop)

- `zip()`
- `range(from, to)`

String

- **Strings are immutable sequences of unicode code points.**
- all sequence operations apply
- Escape characters: `\n, \t, \\, \', \", \\`
- Raw string: `r"This is a string where one backslash is simply one backslash, like this \."`
- Formetted string literals: `f"My name is {name} and I am {age} years old."` , where name and age are variables
- string methods: `capitalize(), casefold(), center(width[, fillchar]), count(sub[, start[, end]]), endswith(suffix[, start[, end]]), expandtabs(tabsize=8), find(sub[, start[, end]]), format(*args, **kwargs), format_map(mapping), index(sub[, start[, end]]), isalnum(), isalpha(), isascii(), isdecimal(), isdigit(), isidentifier(), islower(), isnumeric(), isprintable(), isspace(), istitle(), isupper(), join(iterable), ljust(width[, fillchar]), lower(), lstrip([chars]), partition(sep), replace(old, new[, count]), rfind(sub[, start[, end]]), rindex(sub[, start[, end]]), rjust(width[, fillchar]), rpartition(sep), rsplit([sep[, maxsplit]]), rstrip([chars]), split([sep[, maxsplit]]), splitlines([keepends]), startswith(prefix[, start[, end]]), strip([chars]), swapcase(), title(), upper(), zfill(width)`

Tuple

- **Tuples are immutable sequences of objects.**
- all sequence operations apply
- tuples can also be nested `tup = ((1,2),(3,4))` --> tup[1][0] has value of 3

List

- **Lists are mutable sequences of objects.**
- same as for tuples, all sequence operations apply
- same as tuples, lists can be nested `my_list = [[1,2],[3,4]]` --> `my_list[1][0]` has value of 3
- list elements can be modified `my_list[0] = new_value`
- `list.append()` to add an element to the end on the list
- `list.extend(), delete(), pop([i]), insert(i, x), remove(x), clear(), index(x[, start[, end]]), count(x), sort(key=None, reverse=False), reverse(), copy()`
- `list()` creates a list from an iterable

Dictionary

Dictionary

- mutable **key-value store**, not a sequence (can not slice it)
- initialize: `{}`
- **keys** must be imutable
- **values** can be any data type
- add element `my_dict[new_key] = value`
- modify element `my_dict[key] = new_value`
- check key `if key in my_dict: ...`
- iterate on `keys(), values(), items()`
- Use-cases:
  - mapping
  - counting
  - object properties (more on this today)
- Dictionaries can be combined with lists
- Dictionaries can be stored in json files (and strings) through the `json` module
  - files: `json.dump(), json.load()` , stings: `json.dumps(), json.load()`

Files and Folders

- Opening files: `with open(filename, mode, encoding="utf-8") as f:`
- Modes for opening files
  - Read Only 'r'
  - Write Only 'w'
  - Append Only 'a'
- Within the indented block:
  - `f.write(string)`
  - `f.readline()` returns a string: one line with the \n at the end
  - `for line in f:` iterates over all lines, lines are a string with \n at the end
- Encoding: ASCII and UNICODE
  - `encoding="utf-8"` works in most cases
- filenames and extensions
- folder (directory), path, absolute and relative paths
- text vs. binary files

Modules

- Modules should be imported at the beginning of a python file.
- function (and class) definitions
- Importing modules (at the beginning of the file):
  - `import module` --> we invoke functions by `module.function()`
  - `import module as m` --> we gave the module a short name m, we invoke functions from module `m.function()` # the preserred way

- `from module import function` --> we import just the function *function* from module in the local name space, we use it as `function()`
- `from module import *` --> we import all the content from module in the local name space, we use it as `function()` . This might cause name conflicts, some functions can be *shadowed*.