Python

•••

Basic Introduction

History

- Guido van Rossum
- 1991 (released on Christmas)
- Monty Python and the Flying Circus
- Zen of Python and “Pythonic”

Ecosystem

- Web Frameworks
  - Django, Flask, Sanic, Pyramid, Tornado, etc
- Database
  - SQLite (builtin), PostgreSQL (psycopg2), MySQL (python-mysqldb), etc
- Cloud-related
  - Boto (client for AWS), Lambda (native support), Heroku (native support), etc
- Data Science / ML / AI
  - Jupyter, pandas, tensorflow, etc
- Misc
  - Pytest (test framework), kiwi (mobile development), click (CLI), etc

Installing

- System Python
- Mac Installation
- Linux Installation
- Pyenv

Interpreter, Editors & IDEs

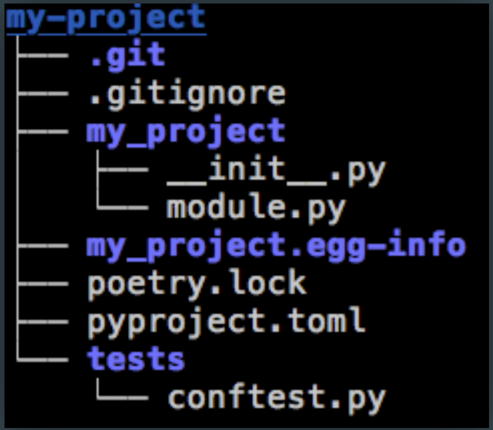
- Interactive Mode
  - iPython (debugger ipdb)
- PyCharm
- Visual Studio Code
- Vim
- Others (Sublime, Textmate, Emacs, ...)

Project Structure

- Virtualenv or Docker
- Pip (and pipx) & PyPI
- setup.{cfg,py}, requirements.txt, pipenv or poetry

Project Structure

- Importing
  - Absolute & Relative imports
  - Namespace
- Standard Python Libraries
- Modules
- Packages
- PYTHONPATH



```
my-project
├── .git
├── .gitignore
├── my_project
│   ├── __init__.py
│   └── module.py
├── my_project.egg-info
├── poetry.lock
├── pyproject.toml
└── tests
    └── conftest.py
```

Programming

Syntax

- Indentation
- Comments
- Documentation
- Coding Style (ok, not syntax but it’s good to learn from the beginning)
    - PEP-8 (eg. trailing comma)
    - flake8, pylint, etc

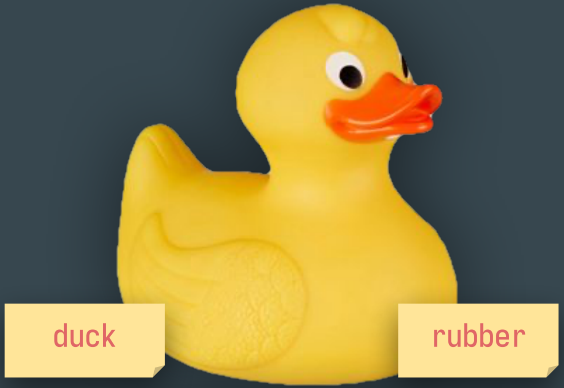
Constant, Operators and Expressions

- Math
  - 2 + 1 (add), 5 - 4 (sub), 2 * 4 (mul), 5 / 2 (div), 5 % 2 (mod), 10**5 (power)
- Binary
  - 2 & 1 (AND) , 7 | 2 (OR), 5 ^ 6 (XOR), ~1 (NOT)
- Logical
  - == (equal), ≠ (not-equal), > (more than), ≥ (more than or equal), < (less than), ≤ (less than or equal)
  - or, and, not, in, not in, is, not is
  - True, False
  - All zero, None or empty collections are “false”
- Constants
  - None (means null and is “false”), ... (ellipsis)

Identifiers and references ("assignment")

- Reference
- Mutability
- Assignment
  - Regular: a = "spam", b = "eggs", c = a
  - Augmented:
    - a += 1, a -= 4, a *= 4, a /= 2, a %= 2
      a **= 5, a &= 1, a |= 2, a ^= 6
    - No "~=", "++", or "--" operations
  - Walrus
    - If a := f(): ...
- Tuple Unpack
  - a, b = 2, 1 # a = 2 and b = 1
  - (a, *b, c, d) = (1, 2, 3, 4, 5, 6) # a=1, b=[2,3,4], c=5 and d=6

optional



```
duck = RubberDuck()   # instantiate object
rubber = duck
```

Builtin Data Types

Numbers

- Integers
  - 1, -5, 0x1f, 0b0001, 0o755, 1_000_000, 10**31532567543…
- Float
  - 1.0, 10**-2, 10e5, 3e-2, 3.5e3
- Complex
  - 5j, 5.5j, -3j
- Boolean
  - True (evals to 1), False (evals to 0)
- Decimals (module decimal)
  - value = Decimal("8.53") # Euros

Strings & Bytes

- Unicode & Codecs
- Strings[🍞, 🏷️, 🔁]
  - `'abc', "abc", '''abc''', """abc"""`
  - `f'Format this {number}'`
  - `'Format {}'.format(123)`
  - `'Format %s style' % ('C',)`
  - `'Escape character is \\ \x20 \N{GRINNING FACE}'`
  - `r'/rege\x/'` (don’t need to scape)
- Bytes[🍞, 🏷️, 🔁]
  - `b"abc", "abc".encode("utf-8")`

🍞 = Sliceable, 🏷️ = Indexable, 🔁 = Iterable

Index

```
>>> s = "Nobody expects the Spanish Inquisition!"
>>> #    ^-- 0                                 ^-- -1
>>>
>>> s[0]
'N'
>>> s[1]
'o'
>>> s[-1]
'!'
>>> s[-2]
'n'
```

🍞 *Slice*

```
>>> s = "Nobody expects the Spanish Inquisition!"
>>> #    ^-- 0              ^-- 19 ^-- 26       ^-- -1
>>>
>>> s[19:]
'Spanish Inquisition!'
>>> s[19:26]
'Spanish'
>>> s[:26]
'Nobody expects the Spanish'
>>> s[:-1] + "."
'Nobody expects the Spanish Inquisition.'
>>> s[:]
'Nobody expects the Spanish Inquisition!'
>>> s[19:-1:2]
'SaihIqiiin'
>>> s[::-1]
'!noitisiuqnI hsinapS eht stcepxe ydoboN'
```

🔁 *Iterate*

```
>>> s = "Nobody expects the Spanish Inquisition!"
>>> #    ^-- 0              ^-- 19 ^-- 26      ^-- -1
>>>
>>> for c in s: print(c, end="-")
...
N-o-b-o-d-y- -e-x-p-e-c-t-s- -t-h-e- -S-p-a-n-i-s-h- -I-n-q-u-i-s-i-t-i-o-n-!-
>>> "".join(c for c in s if c.isalpha())
'NobodyexpectstheSpanishInquisition'
```

Collections

- `List[🍞, 🏷️, 🔁]`
- `Tuple[🍞, 🏷️, 🔁]`
- `Dict[🏷️, 🔁]`
- `Set[🔁]`

🍞 = Sliceable, 🏷️ = Indexable, 🔁 = Iterable

List [🍞, 🏷️, 🔁]

- Syntax:
  - `[0, 1, 2, …]`
  - `list(`*`iterable`*`)`
- Lists are collection objects that can stores different kind of objects and could be changed dynamically

```
>>> l = [0, 1, "two", 3, "four", 5]
>>> l
[0, 1, 'two', 3, 'four', 5]
>>> list("spam eggs!")
['s', 'p', 'a', 'm', ' ', 'e', 'g', 'g', 's', '!']
```

(continue…)

List

```
>>> l = [0, 1, "two", 3, "four", 5]
>>> l[0]  # index!
0
>>> l[-2]
'four'
>>> l[1::2]  # slice!
[1, 3, 5]
>>> l[3] = 'III'  # mutable!
>>> l
[0, 1, 'two', 'III', 'four', 5]
>>> l.append('six')
>>> l.insert(4, '3.5')
>>> l
[0, 1, 'two', 'III', '3.5', 'four', 5, 'six']
>>> del l[4]
>>> l
[0, 1, 'two', 'III', 'four', 5, 'six']
(continue...)
```

List

```
>>> l
[0, 1, 'two', 'III', 'four', 5, 'six']
>>> copy = l[:]  # copy!
>>> copy
[0, 1, 'two', 'III', 'four', 5, 'six']
>>> l[3] = 3
>>> l
[0, 1, 'two', 3, 'four', 5, 'six']
>>> copy
[0, 1, 'two', 'III', 'four', 5, 'six']
>>> l = list("spam eggs!")
>>> l
['s', 'p', 'a', 'm', ' ', 'e', 'g', 'g', 's', '!']
>>> for c in l: print(c.upper(), end='.')  # iterable!
...
S.P.A.M. .E.G.G.S.!.
```

(continue...)

List List Comprehension

- Generate list objects iterating over (mapping) collections
- Allow filtering
- List Comprehension always returns a list() object
- Syntax for List Comprehension:

```
l = [expr for item in collection [if expr]]
```

List List Comprehension

```
>>> l = list(range(10))
>>> l
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> odd_numbers = [n for n in l if n % 2]
>>> odd_numbers
[1, 3, 5, 7, 9]
>>> squares = [n ** 2 for n in l]
>>> squares
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> form_credit_card = "1234..4567-8901sdfl/0000"
>>> cleaned = [d for d in if d.isdigit()]
>>> cleaned = "".join(cleaned)
>>> cleaned
'1234456789010000'
>>> f"{cleaned[:4]} {cleaned[4:8]} {cleaned[8:12]} {cleaned[12:]}"
'1234 4567 8901 0000'
```

Tuple

- Syntax:
  - `(0, 1, 2, …)`
  - `tuple(`*`iterable`*`)`
- Tuples are similar to lists but their are immutable
- Supports indexing, slicing and iteration

```
>>> t = (0, 1, "two", 3, "four", 5)
>>> t
(0, 1, 'two', 3, 'four', 5)
>>> tuple("spam eggs!")
('s', 'p', 'a', 'm', ' ', 'e', 'g', 'g', 's', '!')
```

(continue…)

Tuple

```
>>> t = (0, 1, "two", 3, "four", 5)
>>> t[0]  # index!
0
>>> t[-2]
'four'
>>> t[1::2]  # slice!
(1, 3, 5)
>>> t[3] = 'III'  # immutable!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuple

```
>>> t
(0, 1, 'two', 'III', 'four', 5, 'six')
>>> copy = t[:]  # copy immutable objects does not make any sense
>>> copy
(0, 1, 'two', 'III', 'four', 5, 'six')
>>> id(t) == id(copy)  # no copy!
True
>>> t = tuple("spam eggs!")
>>> t
('s', 'p', 'a', 'm', ' ', 'e', 'g', 'g', 's', '!')
>>> for c in t: print(c.upper(), end='.')  # iterable!
...
S.P.A.M. .E.G.G.S.!.
```

Dict

- Syntax:
  - `{'k1': 'v1', 'k2': 'v2', …}`
  - dict(*map|k/v tuples|**kwargs*)
- Dict are collection objects that can stores different kind of objects. These objects could be recovered by his keys. Dicts could be changed dynamically
- Supports indexing and iteration

```
>>> d = {'key1': 'value1', 'key2': 2, 3: 'value3'}
>>> d
{'key1': 'value1', 'key2': 2, 3: 'value3'}
>>> dict(key1=1, key2=2)
{'key1': 1, 'key2': 2}
>>> dict([('key1', 1), ('key2', 2)])
```

(continue…)

Dict

```
>>> TODO
```

Set

- Syntax:
    - {0, 1, 2, ...} (warning: {} is not an empty set. It is a empty dict)
    - set(*iterable*)
- Set are collection objects that can stores different kind of objects ensuring that they are unique. Sets could be changed dynamically
- Supports iteration

```
>>> s = {1, 2, 3}
>>> s
{1, 2, 3}
>>> s = set("spam eggs!")
>>> s
{'!', 's', ' ', 'a', 'e', 'g', 'p', 'm'}
>>> empty = set()
>>> empty
set()
```

(continue...)

Control Commands

- if *expression* / elif *expression* / else
- while *expression* (else)
- for … in [iterable] (else)

Functions

- Functions are objects
- Calling functions
- Arguments
  - Required, optional (arg=0), args list (*args), kwargs (**kwargs)
- def ...
- lambda
- Scope (global, nonlocal, local)
- Decorators

Exercise: Coding with Tests

- Installing pytest
- Exercise
  - Convert decimal numerals to roman
- Test-driven development
  - Write a test
  - Run test
  - Make it pass
  - Refactor
  - Repeat

Roman numeral (n)	Decimal value (v)
I	1
IV	4
V	5
IX	9
X	10
XL	40
L	50
XC	90
C	100
CD	400
D	500
CM	900
M	1000

Generators

- yield
- Generator Expression

Object-Oriented Programming

- Objects
  - Classes
  - Instances
- Methods
  - Class methods
  - Static methods
- Attributes / Properties
- Inheritance

Object-Oriented Programming

- Magic Methods & Operator overriding
- Pythonic Object Style
    - No cascading methods
    - Methods that changes objects “in-place” returns None
    - Functions instead of static methods
    - No “one file per class”
- Black Magic and Meta Classes

Exception Handling

- Builtin Exceptions (classes)
- try / except / finally / else
- raise / raise from
- Pythonic Exception Style
  - If you don’t know how to handle a exception just leave it unhandled
  - Prefers exceptions to flag return

Missing Parts

- Asynchronous Development
  - await, async, loops, etc
- Tons of Standard Library Modules
- Standard APIs for database (DB-API), Web Gateways (WSGI & ASGI)