# 세상에 나쁜 코드는 없다

## 계산과학자를 위한 코드 잘 짜는 법

서울대학교 정누리

jnooree@snu.ac.kr

# Who am I?

- 서울대 계산생물학 연구실 석박통합과정 (2022-)
- 동 연구실 클러스터 관리자 (2022-)

- 소프트웨어 & DevOps 엔지니어
- Python/C++ 프로그래머
- 오픈소스 기여자 (scipy, e3nn, ruff, homebrew, ...)
- 연구실 공용 코드 메인테이너
- 연구실 내 SWE 세미나 기획 (2022-2025, 5회)

# Where did this talk come from?

- 연구실 코드에서 자주, 공통적으로 보이는 문제점
- 개발 경험에서 얻은 교훈
- 소프트웨어 엔지니어링 세미나 진행 경험

# Why this session?

- 코드에 어떤 문제가 있는지 분석해보기
- 어떤 코드가 좋은 코드인지에 대해 생각해보기
- 여러분이 더 쉽게, 잘 코딩할 수 있도록 돕기

# What will be covered?

- 무엇이 좋은 코드인가?
- 어떻게 하면 좋은 코드를 짤 수 있는가?
- 좋은 코드를 쉽게 짜기 위한 도구 제공

# How?

- 파이썬 코드 예시와 설명으로
- 대부분은 실제 코드베이스에서 직접 가져온 코드
- 질문해주세요!

# What is *code*?

# What is *good* code?

# Part I: Coding Is Communication

Naming, comments, and functions

# What makes code readable?

# What makes code *un*readable?

# What makes code *un*readable?

```
def genTotalResForModel(self,model_sequence,ref_sequence,ulr):
    num_model = model_sequence.size(0)
    ref_1 = ref_sequence[:ulr[0]-1,1:13].reshape(-1,4,3).repeat(num_model,1,1,1)
    ref_2 = ref_sequence[ulr[1]:,1:13].reshape(-1,4,3).repeat(num_model,1,1,1)
    model_sequence = model_sequence[:,:,1:13].reshape(num_model,-1,4,3)
    model_total = torch.cat([ref_1,model_sequence,ref_2],dim=1)

    return model_total
```

# What makes code *un*readable?

```
def genTotalResForModel(self, model_sequence, ref_sequence, ulr):
    num_model = model_sequence.size(0)
    ref_1 = (
        ref_sequence[: ulr[0] - 1, 1:13]
        .reshape(-1, 4, 3)
        .repeat(num_model, 1, 1, 1)
    )
    ref_2 = (
        ref_sequence[ulr[1] :, 1:13]
        .reshape(-1, 4, 3)
        .repeat(num_model, 1, 1, 1)
)
model_sequence = model_sequence[:, :, 1:13].reshape(num_model, -1, 4, 3)
model_total = torch.cat([ref_1, model_sequence, ref_2], dim=1)

return model_total
```

Use **formatters!** 

# What makes code *un*readable?

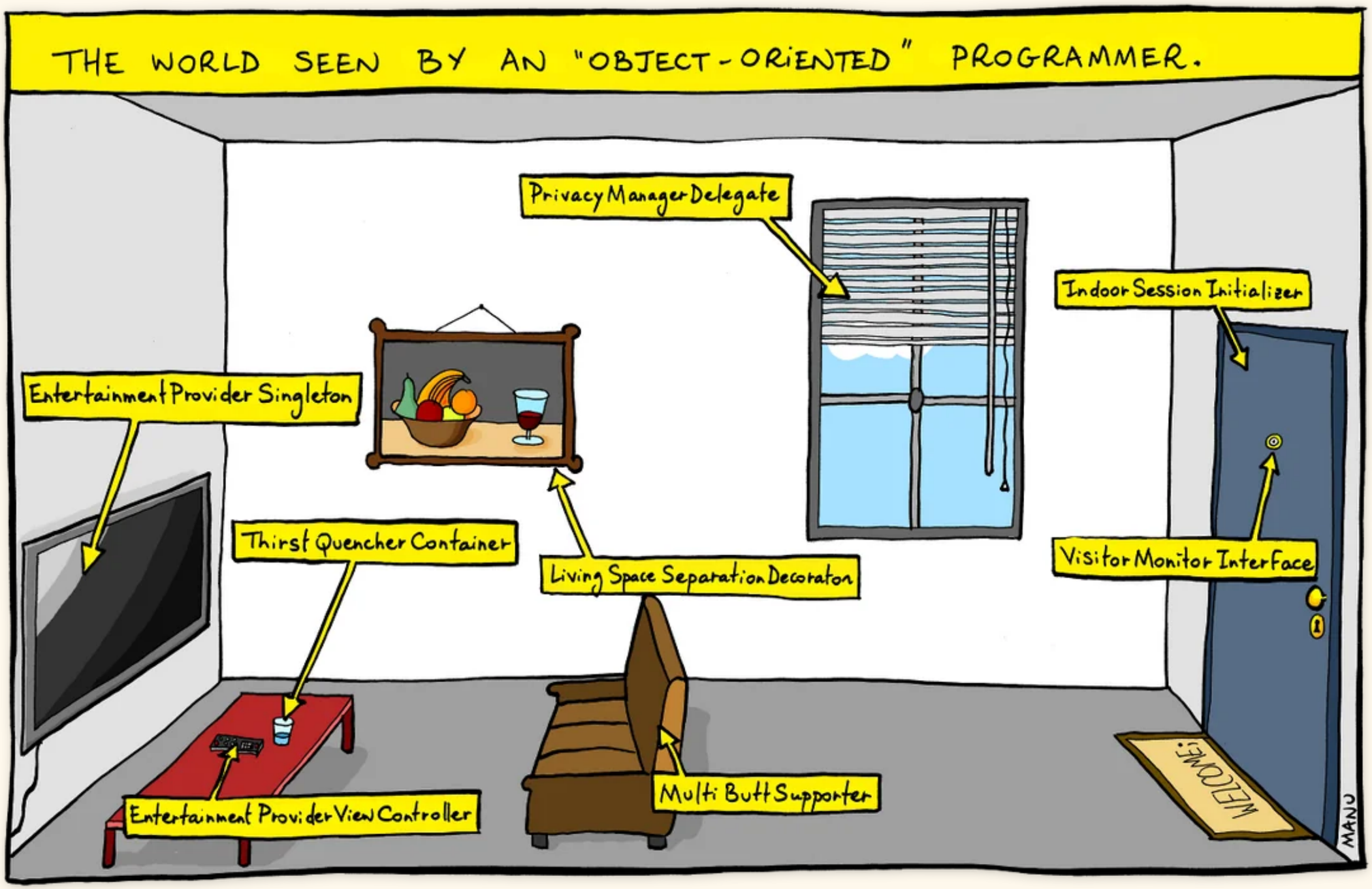
```
b = a(...)
e = b.c(d=1)
```

# Coding is communication

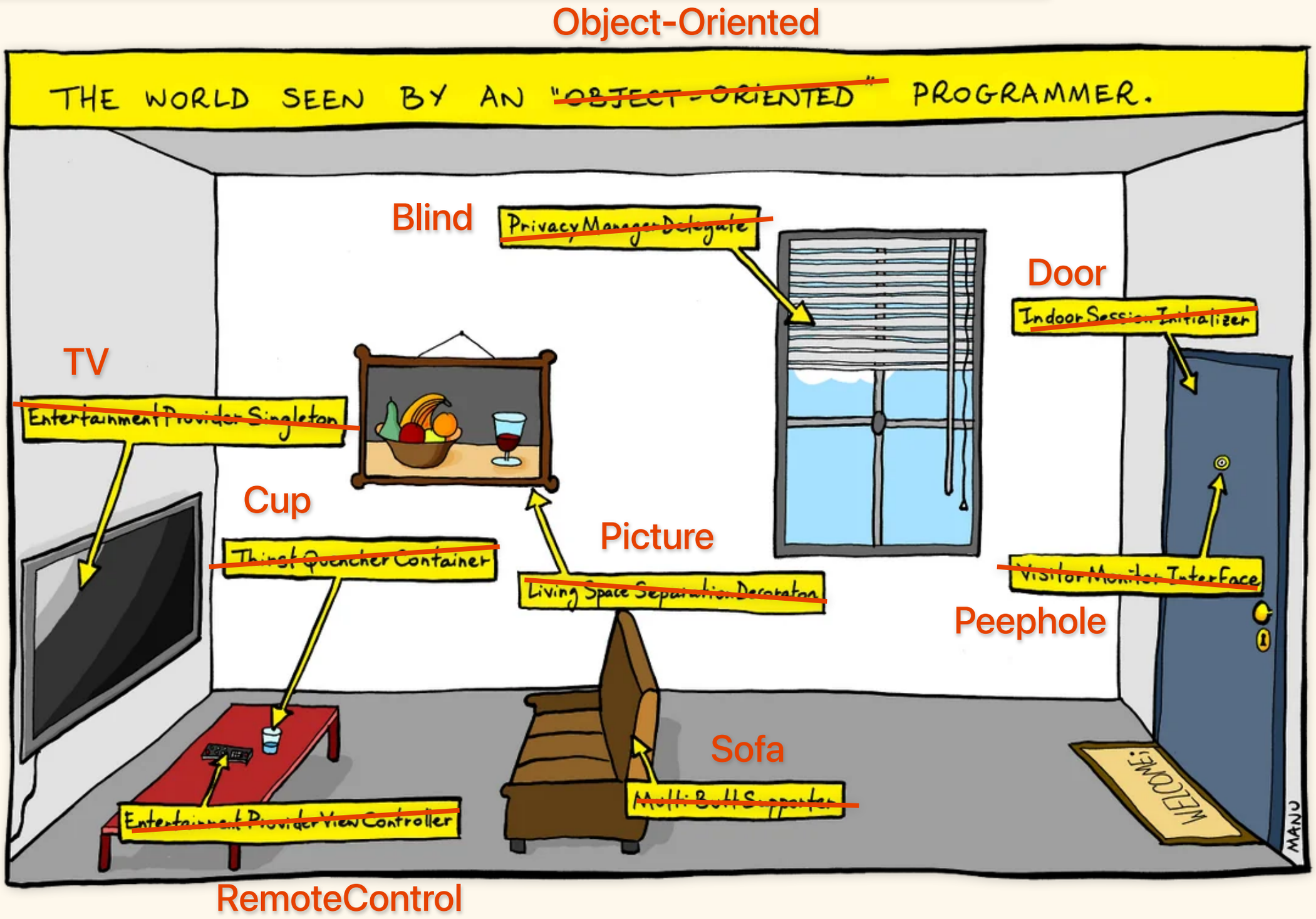
```
b = a(...)
e = b.c(d=1)
```

```
arr = np.array(...)
cntr = arr.mean(axis=1)
```

# Don't be too noisy



# Don't be too noisy



# Don't be too noisy

```
class Point:
    def getX(self): ...
    def getY(self): ...
    def getZ(self): ...
```

# Don't be too noisy

```
class Point:
    def x(self): ...
    def y(self): ...
    def z(self): ...
```

# Don't be too noisy

```
class Point:
    def getX(self): ...
    def setX(self, x): ...

    def getY(self): ...
    def setY(self, y): ...

    def getZ(self): ...
    def setZ(self, z): ...
```

# Don't be too noisy

```
class Point:
    def __init__(self):
        self.x = ...
        self.y = ...
        self.z = ...
```

More on this later

# Let's talk about comments

- **Avoid** comments if possible
- Comment **why**, not **what**

# Don't do this

```
138  # Initialize a job
139  job = Galaxy.initialize(title=title, mkdir=(not fn.cwd))
```

```
74  # Renumber given pocket residues if there are
75  pocket_res = []
76  if pocket_given:
77      tmp_res = []
78      for resNo in fn.residue.split(','):
79          tmp_res.append(int(resNo))
80      pocket_res = renumber_to_new(init.resmap,tmp_res)
```

Taken from GalaxyPipe source code [1] [2]

# Instead, do this

```
def argwhere(a):
    # nonzero does not behave well on 0d, so promote to 1d
    if np.ndim(a) == 0:
        a = shape_base.atleast_1d(a)
        # then remove the added dimension
        return argwhere(a)[:,:0]
    return transpose(nonzero(a))
```

Taken from Numpy source code

# Because comments lie

```
164  # TODO: Reconstrucut M molecule embeddings into N complex graphs
165  batch_complex_graphs = reconstruct_to_complex_graph(
166      batch3d_list,
167      batch_mol_metadata_per_complex,
168      batch_info,
169  )
170  # TODO: Return N complex graphs
171  return batch_complex_graphs
```

Taken from GalaxyMol source code

# Why is this a bad function?

```
def str_to_tensor_decoys(
    pdb_name,
    protein_type,
    pdb_fn: str,
    decoy_list: list,
    len_seq: int,
    get_res_idx,
    target_atom,
):
    tensor_coord = torch.zeros(len(decoy_list), len_seq, MAX_NUM_ATOM, 5)

    r_stat = False
    pdb = open(pdb_fn).readlines()
    n_decoy = 0
    for line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            if int(line.split()[-1]) in decoy_list:
                r_stat = True
                continue
        if not r_stat:
            continue
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue

        res_no = int(line[22:26])
        chain_id = REF_CHAIN.index(line[21])
        if protein_type == "ab_ag":
            if chain_id == 0:
                real_Hchain = pdb_name.split("_")[1]
                chain_id = REF_CHAIN.index(real_Hchain)
        res_idx = get_res_idx[chain_id][res_no]
        atm_name = line[12:16].strip()
        aa = line[17:20]
        aa_type = AA3_IDX[aa]
        tensor_coord[...] = torch.tensor(...)

    return tensor_coord
```

# Why is this a bad function?

```python
def str_to_tensor_decoys(
    pdb_name,
    protein_type,
    pdb_fn: str,
    decoy_list: list,
    len_seq: int,
    get_res_idx,
    target_atom,
):
    tensor_coord = torch.zeros(len(decoy_list), len_seq, MAX_NUM_ATOM, 5)

    r_stat = False
    pdb = open(pdb_fn).readlines()
    n_decoy = 0
    for line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            if int(line.split()[-1]) in decoy_list:
                r_stat = True
                continue
    if not r_stat:
        continue
    if not line.startswith("ATOM"):
        continue
    if line[12:16].strip() not in target_atom:
        continue

    res_no = int(line[22:26])
    chain_id = REF_CHAIN.index(line[21])
    if protein_type == "ab_ag":
        if chain_id == 0:
            real_Hchain = pdb_name.split("_")[1]
            chain_id = REF_CHAIN.index(real_Hchain)
    res_idx = get_res_idx[chain_id][res_no]
    atm_name = line[12:16].strip()
    aa = line[17:20]
    aa_type = AA3_IDX[aa]
    tensor_coord[...] = torch.tensor(...)

return tensor_coord
```

```python
def str_to_tensor_whole(
    pdb_name,
    use_all_atom,
    protein_type,
    pdb_fn: str,
    fa_fn: str,
    ulr_fn: str,
    decoy_pdb_fn: str,
    decoy_list: list,
    decoy_num: int,
):

    target_atom = ...

    fa_s, len_seq = read_fasta(fa_fn)
    pdb = open(pdb_fn).readlines()
    #
    tensor_coord = torch.zeros(len_seq, MAX_NUM_ATOM, 5)
    ...  # other initializations
    #
    res_idx = -1
    res_no = 0
    for line in pdb:
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue
        if int(line[22:26]) != res_no:
            res_idx += 1
        res_no = int(line[22:26])
        atm_name = line[12:16].strip()
        aa = line[17:20]
        aa_type = AA3_IDX[aa]
        if atm_name == "CA":
            ...  # special handling for CA atom
        tensor_coord[...] = torch.tensor(...)
    #
    dic = ...
    return dic
```

```python
def str_to_tensor_wo_reference(
    pdb_name,
    pdb_fn: str,
    fa_fn: str,
    ulr_fn: str,
    decoy_pdb_fn: str,
    decoy_list: list,
    target_atom,
    decoy_num=32,
):
    fa_s, len_seq = read_fasta(fa_fn)
    pdb = open(pdb_fn).readlines()
    #
    tensor_coord = torch.zeros(decoy_num, len_seq, len(target_atom), 3)
    ...  # other initializations
    #
    res_idx = -1
    res_no = 0
    for line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            model_idx = int(line.split()[-1])
            res_idx = -1
            res_no = 0
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue
        if int(line[22:26]) != res_no:
            res_idx += 1
        res_no = int(line[22:26])
        chain_id = REF_CHAIN.index(line[21])
        atm_name = line[12:16].strip()
        if atm_name == "CA":
            ...  # special handling for CA atom
        tensor_coord[...] = torch.tensor(...)

    if len(ulr_fn) == 3:
        ulr_chain, from_res_no, to_res_no = ulr_fn
    else:
        from_res_no, to_res_no, ulr_chain = read_ulr(ulr_fn)
    #
    dic = ...
    return dic
```

# Why is this a bad function?

```python
def str_to_tensor_decoys(
    pdb_name,
    protein_type,
    pdb_fn: str,
    decoy_list: list,
    len_seq: int,
    get_res_idx,
    target_atom,
):
    tensor_coord = torch.zeros(len(decoy_list), len_seq, MAX_NUM_ATOM, 5)

    r_stat = False
    pdb = open(pdb_fn).readlines()
    n_decoy = 0
    for line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            if int(line.split()[-1]) in decoy_list:
                r_stat = True
                continue
        if not r_stat:
            continue
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue

        res_no = int(line[22:26])
        chain_id = REF_CHAIN.index(line[21])
        if protein_type == "ab_ag":
            if chain_id == 0:
                real_Hchain = pdb_name.split("_")[1]
                chain_id = REF_CHAIN.index(real_Hchain)
        res_idx = get_res_idx[chain_id][res_no]
        atm_name = line[12:16].strip()
        aa = line[17:20]
        aa_type = AA3_IDX[aa]
        tensor_coord[...] = torch.tensor(...)

    return tensor_coord
```

```python
def str_to_tensor_whole(
    pdb_name,
    use_all_atom,
    protein_type,
    pdb_fn: str,
    fa_fn: str,
    ulr_fn: str,
    decoy_pdb_fn: str,
    decoy_list: list,
    decoy_num: int,
):
    target_atom = ...

    fa_s, len_seq = read_fasta(fa_fn)
    pdb = open(pdb_fn).readlines()
    #
    tensor_coord = torch.zeros(len_seq, MAX_NUM_ATOM, 5)
    ...  # other initializations
    #
    res_idx = -1
    res_no = 0
    for line in pdb:
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue
        if int(line[22:26]) != res_no:
            res_idx += 1
        res_no = int(line[22:26])
        atm_name = line[12:16].strip()
        aa = line[17:20]
        aa_type = AA3_IDX[aa]
        if atm_name == "CA":
            ...  # special handling for CA atom
        tensor_coord[...] = torch.tensor(...)
    #
    dic = ...
    return dic
```

```python
def str_to_tensor_wo_reference(
    pdb_name,
    pdb_fn: str,
    fa_fn: str,
    ulr_fn: str,
    decoy_pdb_fn: str,
    decoy_list: list,
    target_atom,
    decoy_num=32,
):
    fa_s, len_seq = read_fasta(fa_fn)
    pdb = open(pdb_fn).readlines()
    #
    tensor_coord = torch.zeros(decoy_num, len_seq, len(target_atom), 3)
    ...  # other initializations
    #
    res_idx = -1
    res_no = 0
    for line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            model_idx = int(line.split()[-1])
            res_idx = -1
            res_no = 0
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue
        if int(line[22:26]) != res_no:
            res_idx += 1
        res_no = int(line[22:26])
        chain_id = REF_CHAIN.index(line[21])
        atm_name = line[12:16].strip()
        if atm_name == "CA":
            ...  # special handling for CA atom
        tensor_coord[...] = torch.tensor(...)

    if len(ulr_fn) == 3:
        ulr_chain, from_res_no, to_res_no = ulr_fn
    else:
        from_res_no, to_res_no, ulr_chain = read_ulr(ulr_fn)
    #
    dic = ...
    return dic
```

# Why is this a bad function?

```python
def str_to_tensor_decoys(
    pdb_name,
    protein_type,
    pdb_fn: str,
    decoy_list: list,
    len_seq: int,
    get_res_idx,
    target_atom,
):
    tensor_coord = torch.zeros(len(decoy_list), len_seq, MAX

    r_stat = False
    pdb = open(pdb_fn).readlines()
    n_decoy = 0
    for line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            if int(line.split()[-1]) in decoy_list:
                r_stat = True
                continue
        if not r_stat:
            continue
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue

        res_no = int(line[22:26])
        chain_id = REF_CHAIN.index(line[21])
        if protein_type == "ab_ag":
            if chain_id == 0:
                real_Hchain = pdb_name.split("_")[1]
                chain_id = REF_CHAIN.index(real_Hchain)
        res_idx = get_res_idx[chain_id][res_no]
        atm_name = line[12:16].strip()
        aa = line[17:20]
        aa_type = AA3_IDX[aa]
        tensor_coord[...] = torch.tensor(...)

    return tensor_coord
```

```python
def str_to_tensor_whole(
    pdb_name,
    use_all_atom,
```

```
def str_to_tensor(
    ...,
    decoys=False,
    whole=False,
    reference=True,
):
    ...
```

```python
        atm_name = line[12:16].strip()
        aa = line[17:20]
        aa_type = AA3_IDX[aa]
        if atm_name == "CA":
            ...  # special handling for CA atom
        tensor_coord[...] = torch.tensor(...)
    #
    dic = ...
    return dic
```

```python
def str_to_tensor_wo_reference(
    pdb_name,
    pdb_fn: str,
    fa_fn: str,
    lr_fn: str,
    coy_pdb_fn: str,
    coy_list: list,
    rget_atom,
    coy_num=32,

    _s, len_seq = read_fasta(fa_fn)
    b = open(pdb_fn).readlines()

    nsor_coord = torch.zeros(decoy_num, len_seq, len(target_atom), 3)
    .  # other initializations

    s_idx = -1
    s_no = 0
    r line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            model_idx = int(line.split()[-1])
            res_idx = -1
            res_no = 0
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue
        if int(line[22:26]) != res_no:
            res_idx += 1
        res_no = int(line[22:26])
        chain_id = REF_CHAIN.index(line[21])
        atm_name = line[12:16].strip()
        if atm_name == "CA":
            ...  # special handling for CA atom
        tensor_coord[...] = torch.tensor(...)

    if len(ulr_fn) == 3:
        ulr_chain, from_res_no, to_res_no = ulr_fn
    else:
        from_res_no, to_res_no, ulr_chain = read_ulr(ulr_fn)
    #
    dic = ...
    return dic
```

# Why is this a bad function?

```python
def str_to_tensor_decoys(
    pdb_name,
    protein_type,
    pdb_fn: str,
    decoy_list: list,
    len_seq: int,
    get_res_idx,
    target_atom,
):
    tensor_coord = torch.zeros(len(decoy_list), len_seq, MAX

    r_stat = False
    pdb = open(pdb_fn).readlines()
    n_decoy = 0
    for line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            if int(line.split()[-1]) in decoy_list:
                r_stat = True
                continue
    if not r_stat:
        continue
    if not line.startswith("ATOM"):
        continue
    if line[12:16].strip() not in target_atom:
        continue

    res_no = int(line[22:26])
    chain_id = REF_CHAIN.index(line[21])
    if protein_type == "ab_ag":
        if chain_id == 0:
            real_Hchain = pdb_name.split("_")[1]
            chain_id = REF_CHAIN.index(real_Hchain)
    res_idx = get_res_idx[chain_id][res_no]
    atm_name = line[12:16].strip()
    aa = line[17:20]
    aa_type = AA3_IDX[aa]
    tensor_coord[...] = torch.tensor(...)

return tensor_coord
```

```python
def str_to_tensor_whole(
    pdb_name,
    use_all_atom,
```

```
def str_to_tensor(
    ...,
    de...s=...se,
    whol...lse,
    n...rend...rue,
):
    ...
```

```
    atm_name = line[12:16].strip()
    aa = line[17:20]
    aa_type = AA3_IDX[aa]
    if atm_name == "CA":
        ...  # special handling for CA atom
    tensor_coord[...] = torch.tensor(...)
#
dic = ...
return dic
```

```python
def str_to_tensor_wo_reference(
    pdb_name,
    pdb_fn: str,
    fa_fn: str,
    lr_fn: str,
    coy_pdb_fn: str,
    coy_list: list,
    rget_atom,
    coy_num=32,

    _s, len_seq = read_fasta(fa_fn)
    b = open(pdb_fn).readlines()

    nsor_coord = torch.zeros(decoy_num, len_seq, len(target_atom), 3)
    .  # other initializations

    s_idx = -1
    s_no = 0
    r line in pdb:
        if line.startswith("MODEL"):
            n_decoy += 1
            model_idx = int(line.split()[-1])
            res_idx = -1
            res_no = 0
        if not line.startswith("ATOM"):
            continue
        if line[12:16].strip() not in target_atom:
            continue
        if int(line[22:26]) != res_no:
            res_idx += 1
        res_no = int(line[22:26])
        chain_id = REF_CHAIN.index(line[21])
        atm_name = line[12:16].strip()
        if atm_name == "CA":
            ...  # special handling for CA atom
        tensor_coord[...] = torch.tensor(...)

    if len(ulr_fn) == 3:
        ulr_chain, from_res_no, to_res_no = ulr_fn
    else:
        from_res_no, to_res_no, ulr_chain = read_ulr(ulr_fn)
    #
    dic = ...
    return dic
```

# Extract *real* functions

```python
def load_pdb(...) -> list[Model]:
    ...


def map_residues(str: Model) -> dict[ResId, int]:
    ...


def model_to_tensor(
    str: Model,
    res_map: dict[ResId, int],
) -> Tensor:
    ...
```

If you **can't name** it,
you probably **won't need** it

# What is a function?

- **Reusable** piece of code
- That does **one thing**
- With **some inputs**
- And gives **an output**

# HowTo: Write good functions

- Keep a **single level** of abstraction
- Do not **hard-code** paths/parameters
- Avoid **side effects**

# Keep a single level of abstraction

```
def str_to_tensor_whole(pdb_fn, fa_fn, ...):
    fa_s, len_seq = read_fasta(fa_fn)
    pdb = open(pdb_fn).readlines()
    #
    ...  # initializations
    #
    for line in pdb:
        ...  # reading etc.

    test2, decoy_flag = str_to_tensor_decoys(...)
    if not decoy_flag:
        return False

    dic = ...
    return dic
```

Mid-level

Low-level

High-level

# Keep a single level of abstraction

```
def multi_dot(arrays, *, out=None):
    n = len(arrays)
    if n == 2:
        return dot(arrays[0], arrays[1], out=out)

    arrays = [asanyarray(a) for a in arrays]

    ndim_first, ndim_last = arrays[0].ndim, arrays[-1].ndim
    if arrays[0].ndim == 1:
        arrays[0] = atleast_2d(arrays[0])
    if arrays[-1].ndim == 1:
        arrays[-1] = atleast_2d(arrays[-1]).T
_assert_2d(*arrays)

if n == 3:
    result = _multi_dot_three(arrays[0], arrays[1], arrays[2], out=out)
else:
    order = _multi_dot_matrix_chain_order(arrays)
    result = _multi_dot(arrays, order, 0, n - 1, out=out)

if ndim_first == 1 and ndim_last == 1:
    return result[0, 0]
elif ndim_first == 1 or ndim_last == 1:
    return result.ravel()
else:
    return result
```

Mostly same level

Taken from Numpy source code

# Don't hard-code

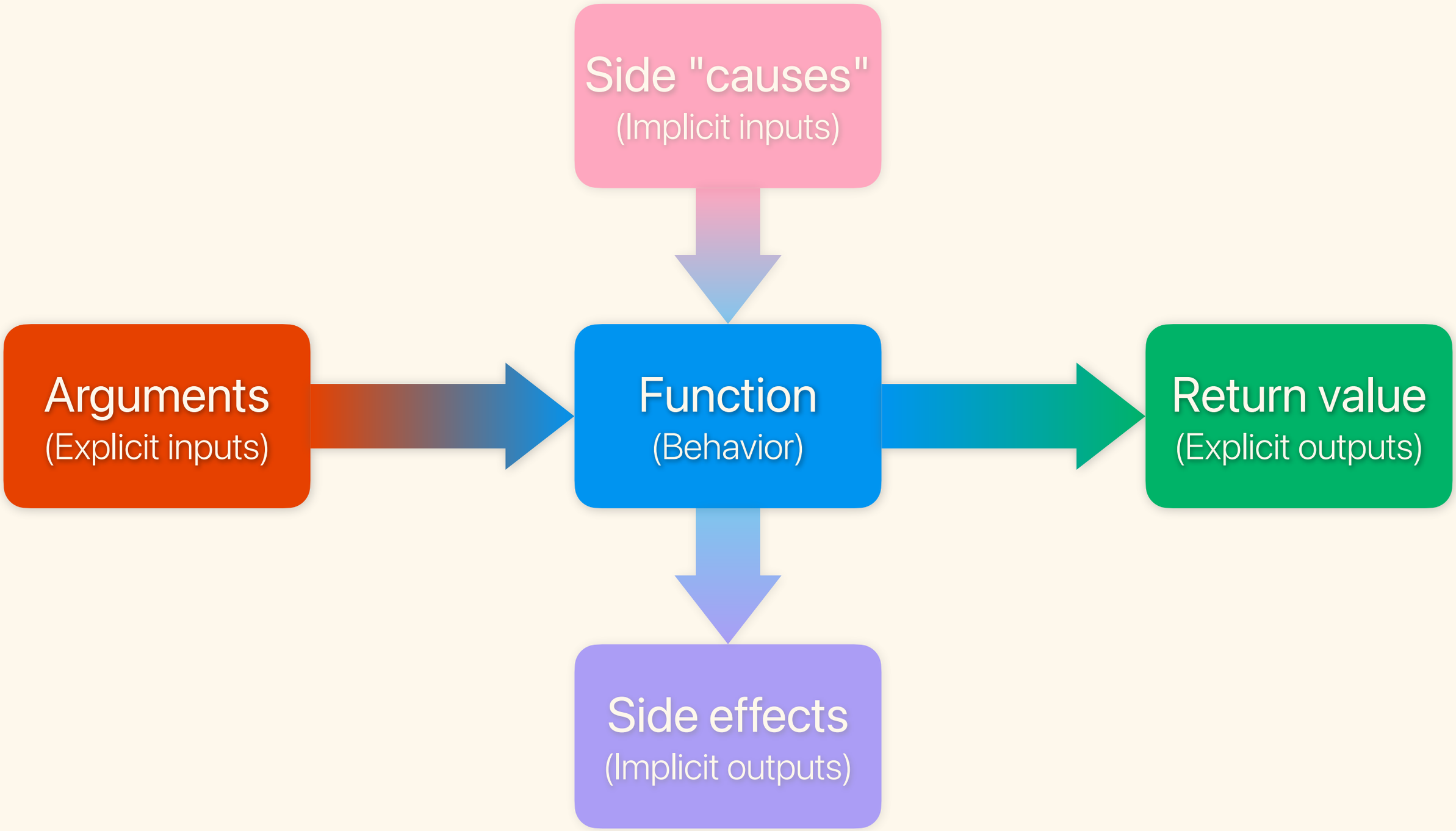
```python
def build_graph(pdb_tag, decoy_type_num, ...):
    num_decoy_group = ...
    group_idx = ...
    graph_s = []
    ...  # complex code filling graph_s
    if decoy_type_num:
        path = f"{GRAPH_PATH}/{pdb_tag}_{decoy_type_num}"
    else:
        path = f"{GRAPH_PATH}/{pdb_tag}"

    if num_decoy_group == 1:
        with open(f"{path}.dat", "wb") as fp:
            print("saved to ", f"{path}.dat")
            pickle.dump(graph_s, fp)
    else:
        with open(f"{path}_{group_idx}.dat", "wb") as fp:
            pickle.dump(graph_s, fp)
```

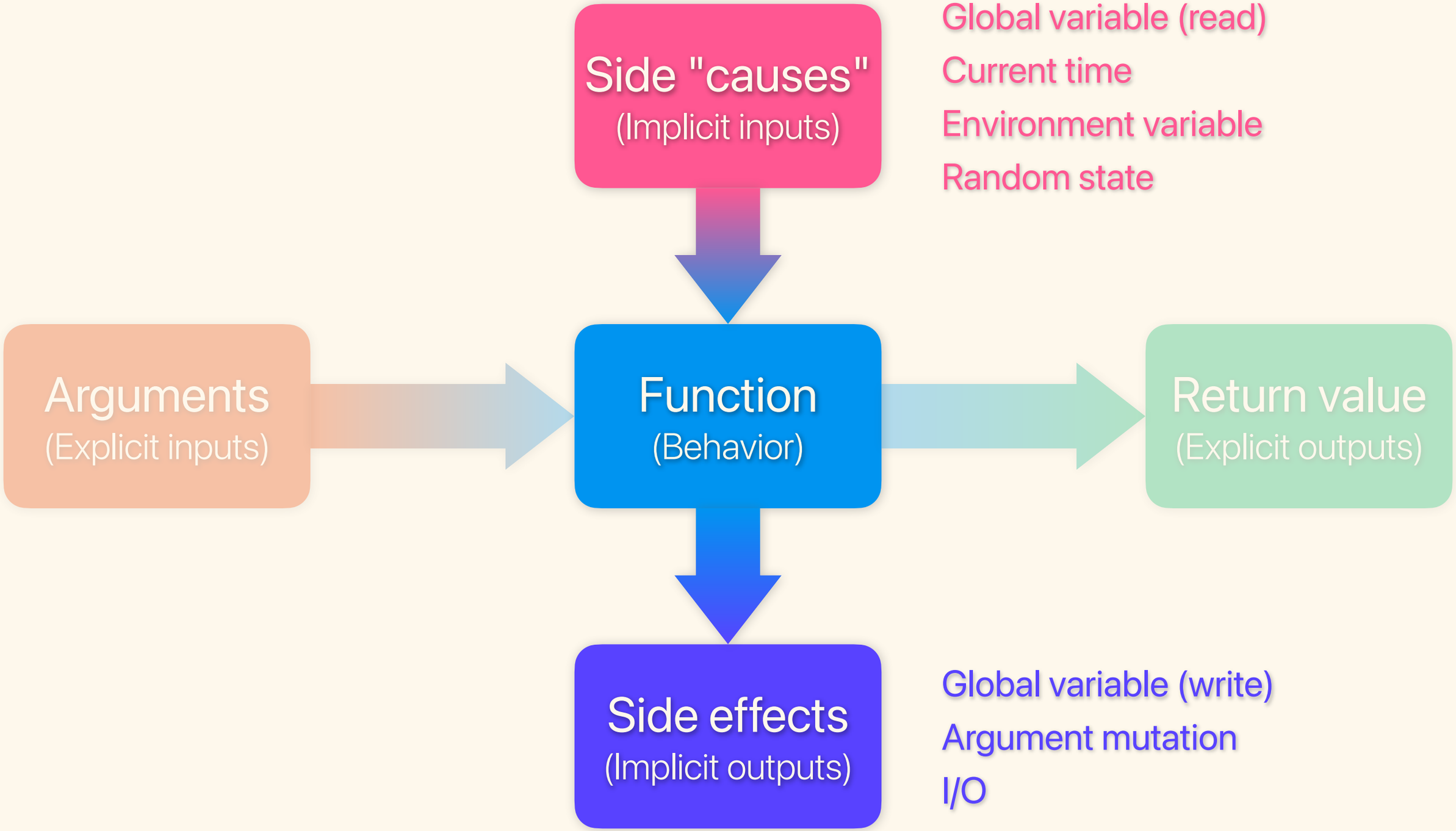
Must copy-paste this to use output

- Just return results
- Pass **as argument**

# Side effects of a function



# Side effects of a function



# Avoid side effects

What is this for?

```
1  prot = PDB()
2  prot.read_pdb(protein_path)
3
4  x_coords = np.array(prot.x_coord)
5  y_coords = np.array(prot.y_coord)
6  z_coords = np.array(prot.z_coord)
7  coords = np.array(prot.R)
```

# Part II: No, You Need More Types

How to *avoid* dictionaries and tuples

# Before starting this section

# Always type hint!

# Do I need a new type?

YES.*

*Unless you write a function with *few* argument(s) and return value(s) and each of which is a *simple* type where the variables carry no more *semantics* than the original types themselves.

# But even for this code?

```
1   def from_smiles(smi: str) -> Mol:
2       ...
```

# But even for this code?

```python
def from_smiles(smi: str) -> Mol:
    ...
```

```
[/Volumes/RAMDisk] python test.py
Traceback (most recent call last):
  File "/Volumes/RAMDisk/test.py", line 9, in <module>
    from_smiles("CC=C")
    ~~~~~~~~~~~^^^^^^^^
  File "/Volumes/RAMDisk/test.py", line 5, in from_smiles
    open(smi)
    ~~~~^^^^^
FileNotFoundError: [Errno 2] No such file or directory: 'CC=C'
```

# But even for this code?

```
1  def from_smiles(smi: str) -> Mol:
2      ...
```

```
1  def from_smiles(smi: Path) -> Mol:
2      ...
```

**Types** carry **semantics**

# Type?

A **programming language** is a system of notation for writing computer programs.[1] Programming languages are described in terms of their syntax (form) and semantics (meaning), usually defined by a formal language. Languages usually provide features such as a type system, variables, and mechanisms for error handling. An implementation of a programming language is required in order to execute programs, namely an interpreter or a compiler. An interpreter directly executes the source code, while a compiler produces an executable program.

# Effect of a dedicated return type

```
561  def get_cath() -> (
562      dict[str, dict[str, dict[str, int | dict[str, list[tuple[str, str]]]]]]
563  ):
564      ...
```

```
137  def get_cath(
138      cath_all: Path, cath_clusters: Path
139  ) -> dict[str, dict[str, list[CathRawDomain]]]:
140      ...
```

```
 28  class CathRawDomain(BaseModel):
 29      id: int
 30      ranges: list[tuple[str, str]]
```

Credit: GalaxyMol source code [1] [2]

# Effect of a dedicated return type

```python
def nurikit_get_infos(
    id1: str,
) -> tuple[dict[str, list[str]], dict[str, dict[str, np.ndarray]], dict[str, list[str]]]:
    ...
```

```python
def nurikit_get_infos(
    pdb_id: str, pdb_orig: Path, tmpd: Path
) -> dict[str, ChainInfo]:
    ...
```

```python
class ChainInfo:
    seq_pieces: list[str]
    resnum_atom: list[str]
    coords: BackboneCoords
```

```python
class BackboneCoords:
    N: list[np.ndarray]
    CA: list[np.ndarray]
    C: list[np.ndarray]
```

Credit: GalaxyMol source code [1] [2]

# Effect of a dedicated input type

```python
def match_number(seq_res: dict[str, str], seq_pieces: dict[str, list[str]],
                 resnum_atom: dict[str, list[str]], m_c: str):
    ...
```

```python
def match_number(seqres: str, chain_info: ChainInfo):
    ...
```

```python
class BackboneCoords:
    N: list[np.ndarray]
    CA: list[np.ndarray]
    C: list[np.ndarray]
```

```python
class ChainInfo:
    seq_pieces: list[str]
    resnum_atom: list[str]
    coords: BackboneCoords
```

Credit: GalaxyMol source code [1] [2]

# Why new types?

- **Self-documenting** code (= readability)
- **Easier change** propagation
  (= maintainability, with tools like Pylance)
- Rigorous static **type analysis** (= robustness)
- Better **completion** & shorter code (= DevEx)

# HowTo: Type creation rule of thumbs

- No more than **3 input arguments** (excluding self)
- **One or two** return values (avoid zero)
- If **cannot type-hint**, create a new type

# ProTip: Use @dataclass!

```python
class Data:
    def __init__(self, a: int, b: str, c: bool):
        self.a = a
        self.b = b
        self.c = c
```

```python
from dataclasses import dataclass

@dataclass
class Data:
    a: int
    b: str
    c: bool
```

# Pitfalls: *abusing* builtin types

## Don't:

```
43  self.labels: list[float | int]
```

## Do:

```
34  class BackboneCoords:
35      N: list[np.ndarray]
36      CA: list[np.ndarray]
37      C: list[np.ndarray]
```

**List** is for:

- Homogeneous,
- Variable-length,
- Indexable,
- Collection of values.

Credit: GalaxyMol source code

# Pitfalls: *abusing* builtin types

## Don't:

```
561  def get_cath() -> (
562      dict[str, dict[str, dict[str, int ¦ dict[str, list[tuple[str, str]]]]]]
563  ):
564      ...
```

## Do:

```
225  def nurikit_get_infos(
226      pdb_id: str, pdb_orig: Path, tmpd: Path
227  ) -> dict[str, ChainInfo]:
228      ...
```

**Dictionary** is for:

- Homogeneous,
- Dynamic,
- Key-value pairs,
- Require key-based lookup.

# Pitfalls: *abusing* builtin types

## Don't:

```python
def recon_targets(
    self, mol: Chem.Mol
) -> list[tuple[tuple[int | float | list[int], DataType, int]]]:
```

## Do:

```python
class CathRawDomain(BaseModel):
    id: int
    ranges: list[tuple[str, str]]
```

**Tuple** is for:

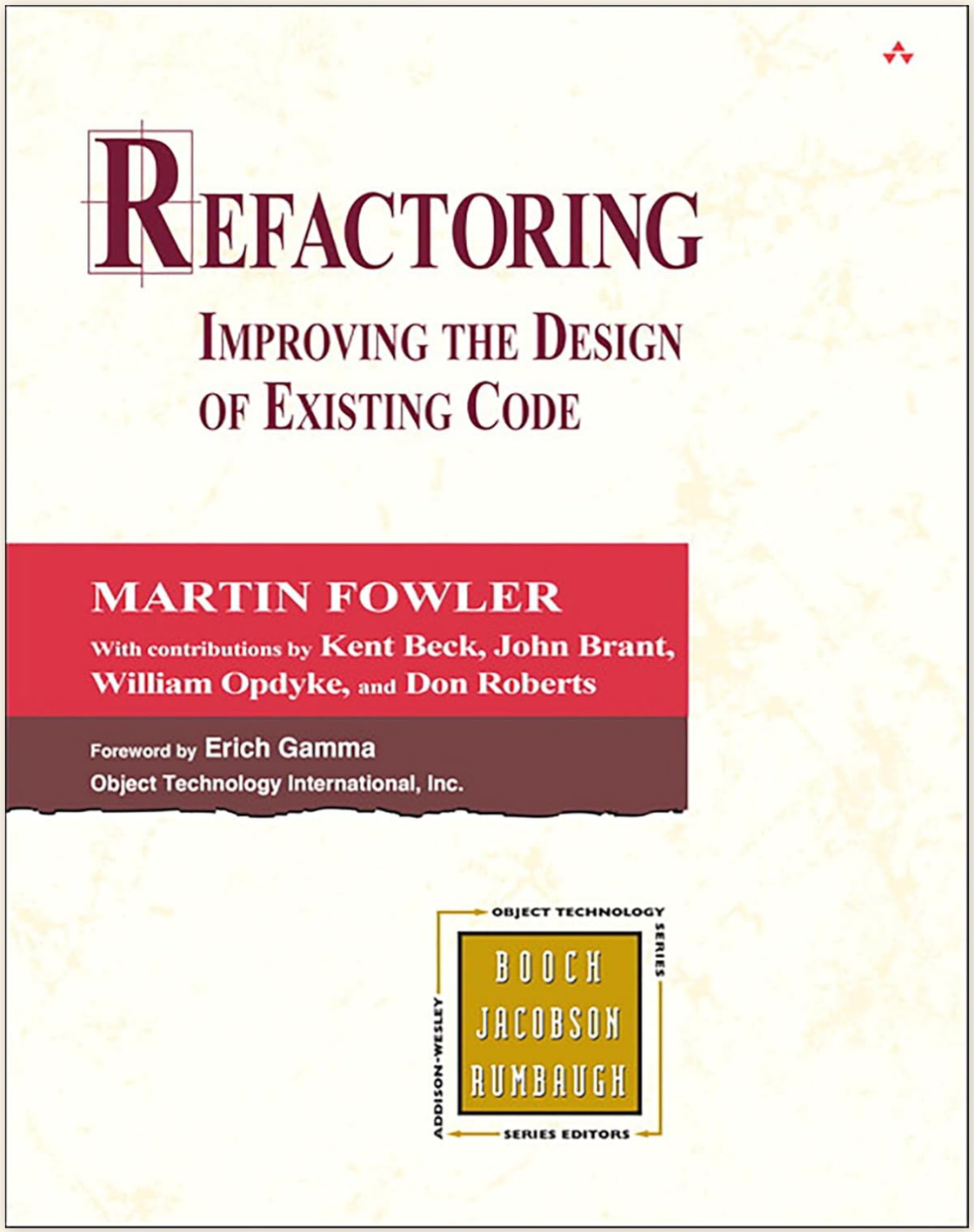
- Short,
- Fixed-type and length,
- Group of values,
- With emergent semantics.

Credit: GalaxyMol source code

# Part III: Code Never Dies

Type design: abstraction, polymorphism, and SOLID principles

# We are focusing on code *refactoring*



# When do you *refactor*?

- Ctrl + C/V (**Redundancy**)
- **Change propagation**
- **Unreadable** and **buggy** code

# When do you *refactor*?

Software engineering

# Software engineering?

*Any problem in computer science*
*can be solved with another* ***level of indirection****.*

– Wheeler, D. J., FRS (emphasis mine)

Often referred to as the *Fundamental Theorem of Software Engineering* (FTSE)

# The most important concept

# Separation of Concerns

# Leaky abstraction creates friction

```
18  if isinstance(pdb, Galaxy.core.PDB):
19      opt_s['infile_pdb'] = pdb.pdb_fn.relpath()
20  elif isinstance(pdb, Galaxy.core.FilePath):
21      opt_s['infile_pdb'] = pdb.relpath()
22  elif isinstance(pdb, str):
23      pdb_fn = Galaxy.core.FilePath(pdb)
24      opt_s['infile_pdb'] = pdb_fn.relpath()
```

What if we **update existing** or **add new** code?

Credit: GalaxyPipe source code

# We need an *interface*

```
18  if isinstance(pdb, Galaxy.core.PDB):
19      opt_s['infile_pdb'] = pdb.pdb_fn.relpath()
20  elif isinstance(pdb, Galaxy.core.FilePath):
21      opt_s['infile_pdb'] = pdb.relpath()
22  elif isinstance(pdb, str):
23      pdb_fn = Galaxy.core.FilePath(pdb)
24      opt_s['infile_pdb'] = pdb_fn.relpath()
```

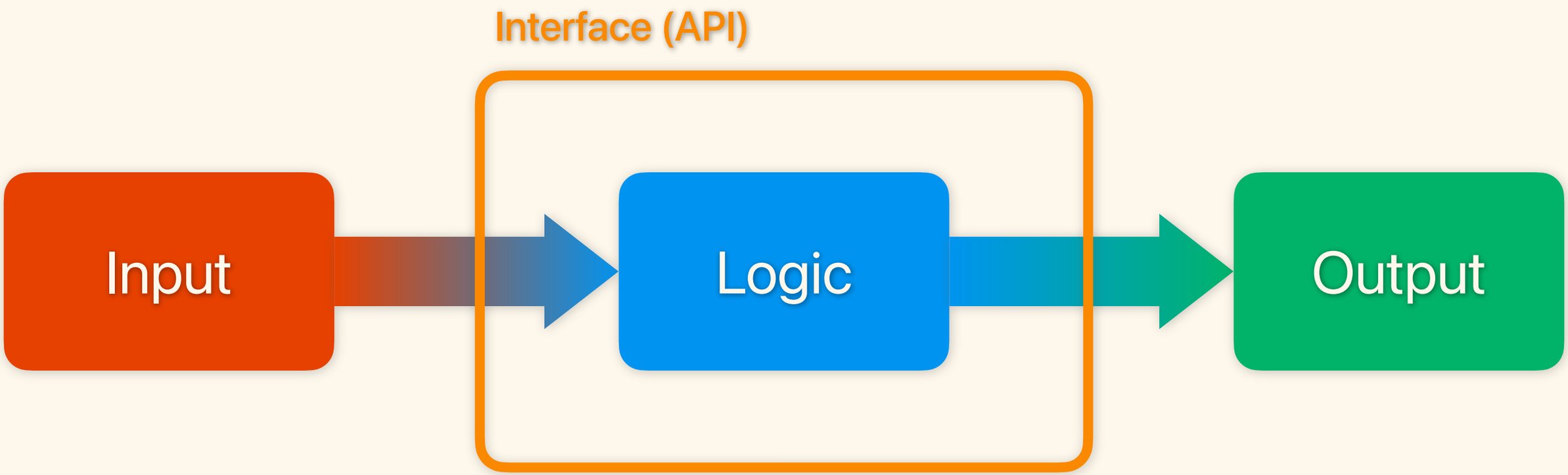
```
18  opt_s['infile_pdb'] = os.fspath(pdb)
```

The **principles** behind this example are:

- ***Encapsulation***, internal details are hidden
- ***Polymorphism***, concrete types are interchangeable

# Any code has input and output

# Interface is abstraction exposed



# Object-oriented programming (≈class)

**The Feynman Problem Solving Algorithm:**

**1. Write down the problem.**

**|**

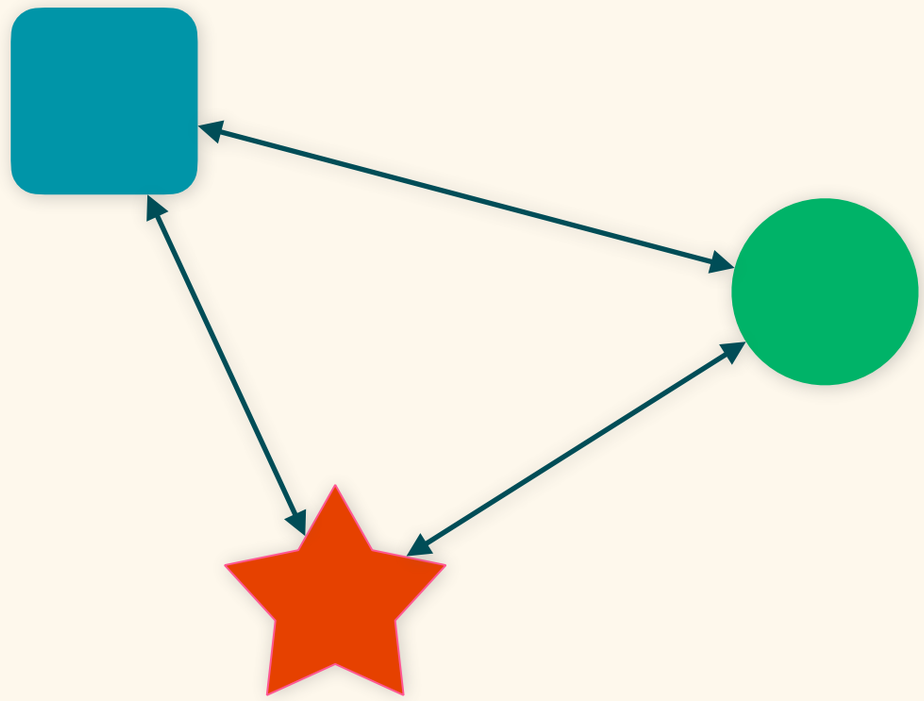
**V**

**2. Think real hard.**

**|**

**V**

**3. Write down the solution.**



Procedural Programming

Object-Oriented Programming

# Interface design with OOP

```
# Superclass; provides an interface
class Module:
    def __call__(self, ...):
        self.forward(...)
```

```
# Subclass; implements the interface
class Linear(Module):
    def forward(self, ...): ...

# All "Modules" can be used similarly
h = model(...)
```

# Pitfalls in class design

- **Avoid logic in initializer** (use factories, pass dependencies)
- **Prefer functions** over methods

# Avoid logic in initializer

```python
class DomFrameFeature:
  def __init__(self, id_ch):
    self.id_ch = id_ch
    data_path = f"{base_path_frame}/{id_ch}.pkl"
    with open(data_path, "rb") as f:
      self.features = pickle.load(f)
    self.rots = self.features["R"]
    self.trs = self.features["T"]
    self.doms = [int(i) for i in self.features["domain_index"]]
```

Provides "shortcut"

```python
# Loading code
feat = DomFrameFeature("example_id")
```

Credit: GalaxyMol source code [1] [2]

# Avoid logic in initializer

```python
class DomFrameFeature:
  def __init__(self, id_ch):
    self.id_ch = id_ch
    data_path = f"{base_path_frame}/{id_ch}.pkl"
    with open(data_path, "rb") as f:
      self.features = pickle.load(f)
    self.rots = self.features["R"]
    self.trs = self.features["T"]
    self.doms = [int(i) for i in self.features["domain_index"]]
```

```python
# Loading code
feat = DomFrameFeature("example_id")
```

```python
class DomFrameFeature:
  def __init__(self, rots, trs, doms):
    self.rots = rots
    self.trs = trs
    self.doms = doms

  @classmethod
  def load_id_ch(cls, id_ch):
    data_path = f"{base_path_frame}/{id_ch}.pkl"
    with open(data_path, "rb") as f:
      features = pickle.load(f)
    return cls(
      rots=features["R"],
      trs=features["T"],
      doms=[int(i) for i in features["domain_index"]],
    )
```

```python
# Loading code
feat = DomFrameFeature.load_id_ch("example_id")
# Now can create directly
feat = DomFrameFeature(rots=..., trs=..., doms=...)
```

# Use factories

Credit: GalaxyMol source code [1] [2]

# Avoid logic in initializer

```python
def __init__(
        self,
        frag_decompose_rule: str = "brics",
        pse_vocab: Optional[Path] = None,
    ):
        if frag_decompose_rule not in ["brics", "pse"]:
            raise NotImplementedError(
                f"{frag_decompose_rule} is not implemented!"
            )
    if frag_decompose_rule == "pse":
        assert pse_vocab is not None, "For PSE, Path of vocab is required."

    self.frag_decompose_rule = frag_decompose_rule
    self.pse_vocab = pse_vocab
```

Subobject initialization or
Implementation selection

Credit: GalaxyMol source code [1] [2]

# Avoid logic in initializer

```python
def __init__(
        self,
        frag_decompose_rule: str = "brics",
        pse_vocab: Optional[Path] = None,
    ):
        if frag_decompose_rule not in ["brics", "pse"]:
            raise NotImplementedError(
                f"{frag_decompose_rule} is not implemented!"
            )
    if frag_decompose_rule == "pse":
        assert pse_vocab is not None, "For PSE, Path of vocab is required."

    self.frag_decompose_rule = frag_decompose_rule
    self.pse_vocab = pse_vocab
```

```python
def __init__(
        self,
        frag_decomposer: Decomposer = BricsDecomposer(),
        pse_vocab: Optional[Path] = None,
    ):
        self.frag_decomposer = frag_decomposer
        self.pse_vocab = pse_vocab
```

## Inject dependencies directly

(*Dependency inversion principle* of SOLID)

Credit: GalaxyMol source code [1] [2]

# Prefer functions over methods

```python
def transform(self):
    x = dict()
    for i, dom in enumerate(self.doms):
        x[dom] = np.concatenate(
            [self.rots[i].reshape(9), self.trs[i].reshape(3)], axis=-1
        )
    return x
```

**Method** should match the **object's behavior**

```python
def transform(doms, rots, trs):
    x = dict()
    for i, dom in enumerate(doms):
        x[dom] = np.concatenate(
            [rots[i].reshape(9), trs[i].reshape(3)], axis=-1
        )
    return x
```

Credit: GalaxyMol source code

# Recap: FTSE

*Any problem in computer science*
*can be solved with another* ***level of indirection***.

– Wheeler, D. J., FRS (emphasis mine)

# Recap: FTSE + α

*Any problem in computer science*
*can be solved with another level of indirection...*
*except for the problem of* ***too many levels of indirection***

# Recap: what is good code?

- Readable
- Working
- Simple
- Maintainable
- Extensible & reusable
- Self-documenting

From a survey made by Intent HQ (2015)

# Recap: what is good code?

- Readable
- Working
- Simple
- Maintainable
- Extensible & reusable
- Self-documenting

From a survey made by Intent HQ (2015)

# OOP in real world

## What should have been done

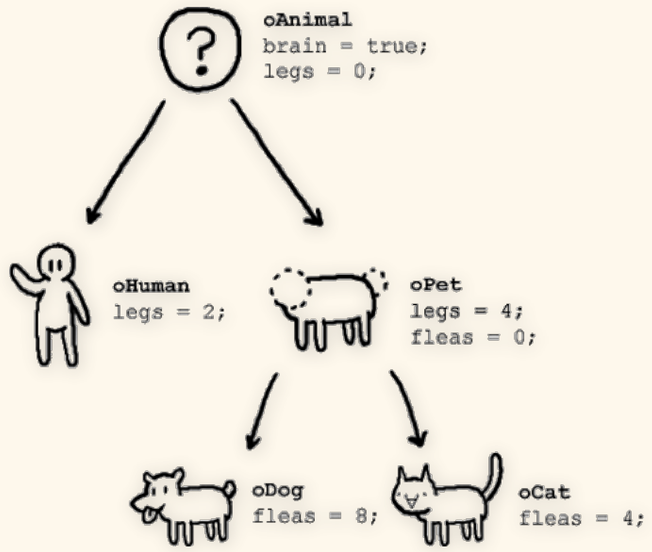



Figure from a Reddit thread

# OOP in real world

## What actually happens

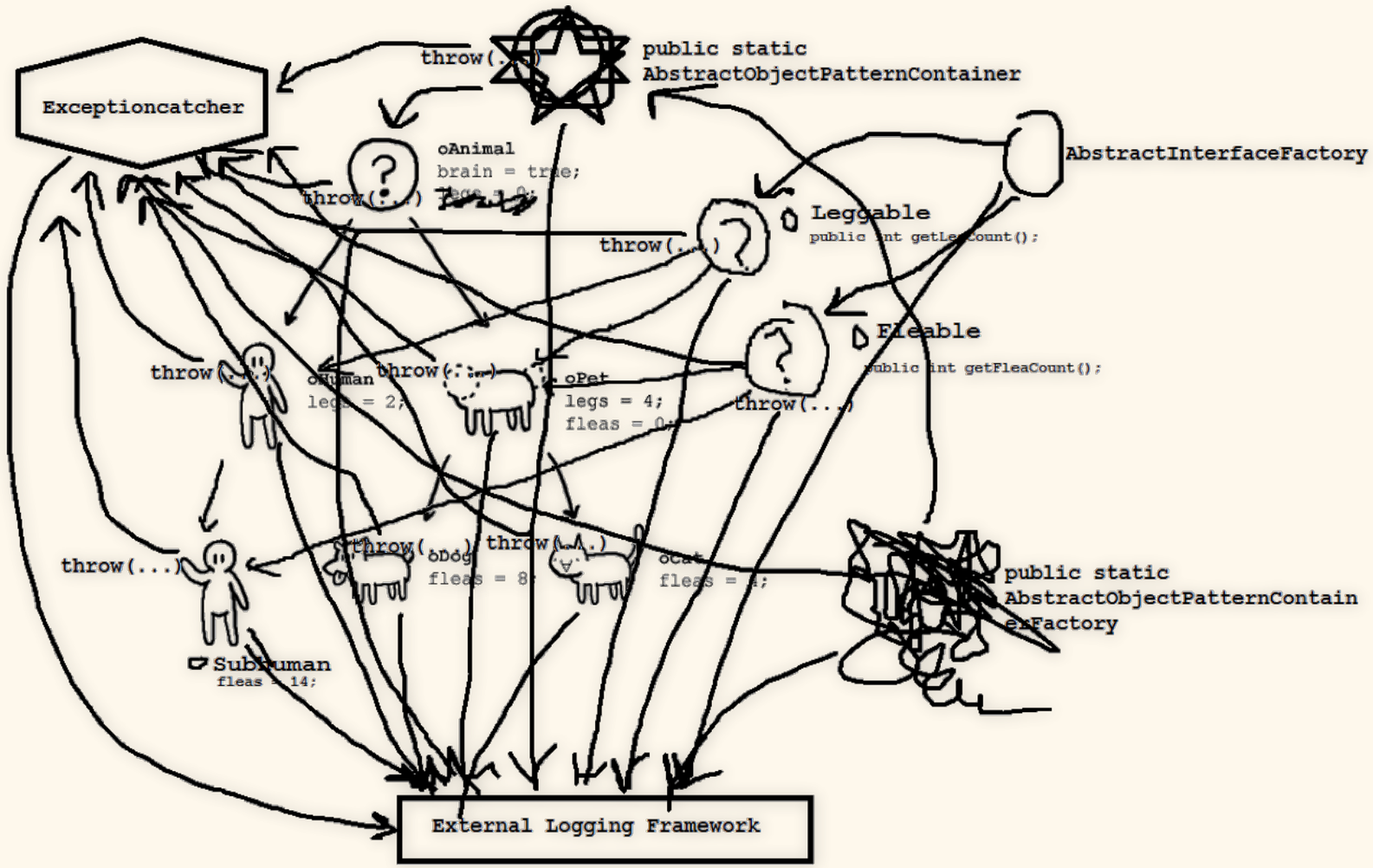
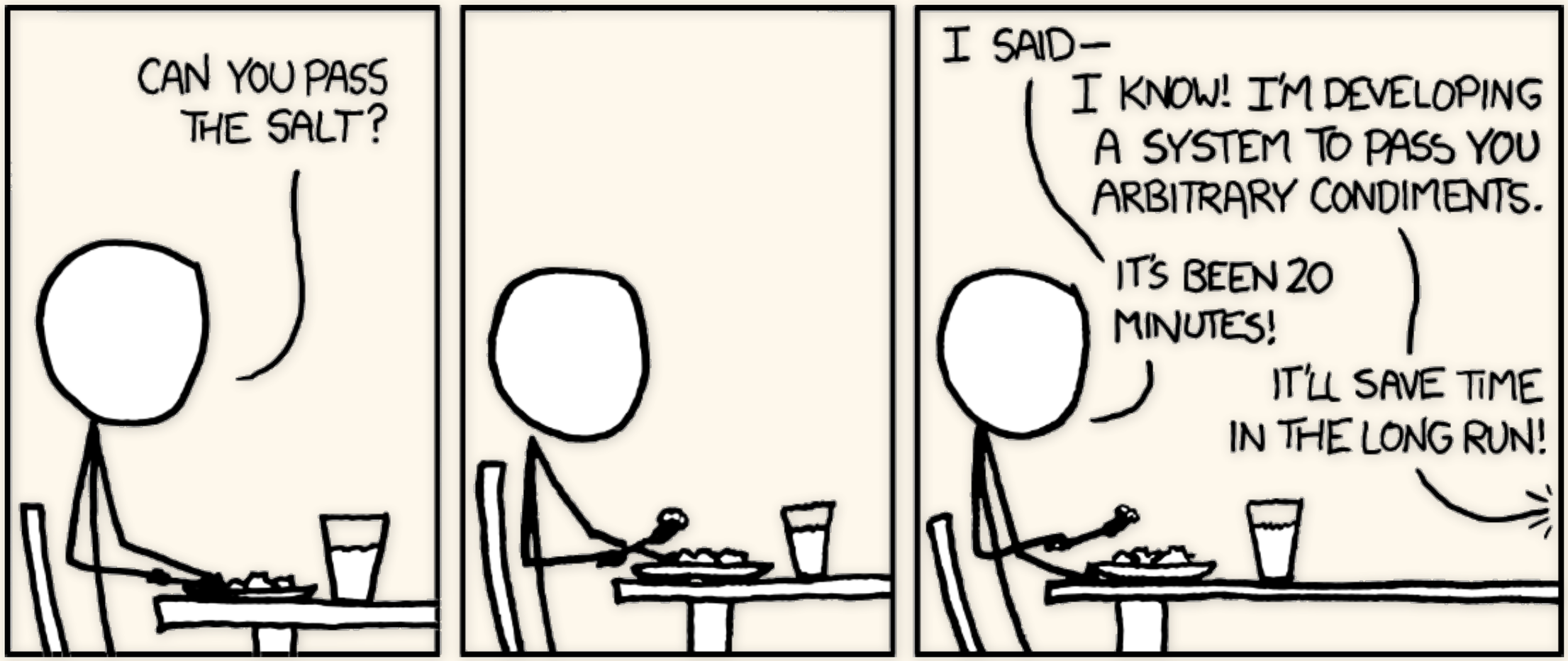



Figure from a Reddit thread

# OOP rule of thumbs

- Reach for OOP *only* to **design interfaces**
- Inherit *only* to **implement an interface**

# Final notes




**Abstraction isn't a plan**; it's a **pattern you discover**

Original figure from xkcd, upscaled with waifu2x

# Part IV:
# Exploring the Python Standard Library

The python you might not know

# Syntax you might not know

- Triple quotes ('docstrings')
- f-strings (formatted string literals) (since 3.6)
- Ternary operator
- `or` as a conditional statement
- Comprehensions
- Assignment expression (since 3.8)
- Loop-else construct

Interested? See the python docs

# Triple quotes

```
>>> long_string = "A very long string with\n" \
... + "multiple line breaks\nand many words"
>>> print(long_string)
A very long string with
multiple line breaks
and many words
```

# Triple quotes

```
>>> long_string = """A very long string with
... multiple line breaks
... and many words"""
>>> print(long_string)
A very long string with
multiple line breaks
and many words
```

# Triple quotes

```
>>> long_string = """
... A very long string with
... multiple line breaks
... and many words"""
>>> print(long_string)

A very long string with
multiple line breaks
and many words
```

Available in all 3.x series

# Triple quotes

```
>>> long_string = """\
... A very long string with
... multiple line breaks
... and many words"""
>>> print(long_string)
A very long string with
multiple line breaks
and many words
```

Available in all 3.x series

# Triple quotes

```
>>> long_string = """\
... A very long string with
...     multiple line breaks
...     and many words"""
>>> print(long_string)
A very long string with
    multiple line breaks
    and many words
```

Available in all 3.x series

# f-strings

```
>>> name = "Nuri Jung"
>>> print(f"My name is {name}")
My name is Nuri Jung
```

Available since 3.6 (Dec 2016)

# f-strings

```
>>> name = "Nuri Jung"
>>> intro = f"My name is {name}"
>>> name = "Foo Bar"
>>> print(intro)
My name is Nuri Jung
```

Available since 3.6 (Dec 2016)

# f-strings

```
>>> result = [0, 1, 2, 3, 4]
>>> print(f"The result is {result}")
The result is [0, 1, 2, 3, 4]
```

Available since 3.6 (Dec 2016)

# f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num}")
The number is 33.333333333333336
```

Available since 3.6 (Dec 2016)

# f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num:.3}")
The number is 33.3
              1 2  3
```

Available since 3.6 (Dec 2016)

Details available here

# f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num:5.3}")
The number is  33.3
              12345
```

Available since 3.6 (Dec 2016)

Details available here

# f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num:05.3}")
The number is 033.3
              12345
```

Available since 3.6 (Dec 2016)

Details available here

# f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num:03.3}")
The number is 33.3
              1 2 3 4
```

Available since 3.6 (Dec 2016)

Details available here

# f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num:.3f}")
The number is 33.333
              1 2 3
```

Available since 3.6 (Dec 2016)

Details available here

# f-strings

```
>>> num = 100 / 3
>>> print(f"{num = :.3f}")
num = 33.333
```

Available since 3.8 (Oct 2019)

Details available here

# Triple-quoted f-strings

```
210             if today_meeting:
211                 message = ""
212             else:
213                 message = """\
214 이번 주 랩미팅은 쉽니다.
215 앞으로 다가올 랩미팅 안내입니다.
216
217 """
218             message += f"""\
219 *[{scheduled.year}년도 제{nth}차 온라인 그룹미팅]*
220 :spiral_calendar_pad: {scheduled.strftime("%m-%d (%a)")} \
221 <!date^{int(scheduled.timestamp())}^{{time}}¦{scheduled.strftime("%H:%M")}>
222 • 진행: {_format_name(chairman)}
223 • 연구발표: {_format_name(speaker)}
224 • 저널발표: {_format_name(journal)}
225 • 저널선택: {_format_name(selector)}
226 • Zoom 링크: <{url}>"""
```

Taken from the galaxybot source code

# Ternary operator

```
if a >= 0:
    print(a)
else:
    print(-a)
```

# Ternary operator

```
print(a if a >= 0 else -a)
```

Available in all 3.x series

# Ternary operator

cf) Equivalent to `(condition) ? x : y` in C-like languages

```
x if (condition) else y
```

Available in all 3.x series

# Another ternary operator?

```
if a:
    foo = a
else:
    foo = b
```

# Another ternary operator?

```
foo = a if a else b
```

# `or` as a conditional statement

```
foo = a or b
```

# `or` as a conditional statement

```
23  _nproc *= int(
24      _os.getenv("SLURM_NTASKS") or _os.getenv("SLURM_NNODES") or 1)
```

Taken from the GalaxyPipe source code

# Comprehensions

```
squares = [x * x for x in range(10)]
```

Is equivalent to

```
squares = []
for x in range(10):
    squares.append(x * x)
```

# Comprehensions

```
squares = [x * x for x in range(10)
                 if x % 2 == 0]
```

Is equivalent to

```
squares = []
for x in range(10):
    if x % 2 == 0:
        squares.append(x * x)
```

# Comprehensions

```
squares = {x: x * x for x in range(10)
                    if x % 2 == 0}
```

Is equivalent to

```
squares = {}
for x in range(10):
    if x % 2 == 0:
        squares[x] = x * x
```

# Assignment expressions

```
while True:
    flag = foo()
    if not flag:
        break
do_something(flag)
```

# Assignment expressions

```
# SyntaxError
while (flag = foo()):
    do_something(flag)
```

# Assignment expressions

```
while (flag := foo()):
    do_something(flag)
```

Available since 3.8 (Oct 2019)

Parenthesis are mandatory

# Assignment expressions

```
160             for meeting in meetings:
161                 sublist = f"""\
162 • *{meeting[0]}*
163       - channel: <#{meeting[1]}>
164       - on: {meeting[2]} {meeting[3]}"""
165                 if (url := meeting[4]):
166                     sublist += f"""
167       - link: <{url}>"""
```

Taken from the galaxybot source code

# Loop-else construct

```
for i in range(10):
    if i == 42:
        break
else:
    print("No answer")
```

Available in all 3.x series

# Loop-else construct

```
for i in range(10):
    if i == 42:
        break
else:
    print("No answer")
```

```
found = False
for i in range(10):
    if i == 42:
        found = True
        break
if not found:
    print("No answer")
```

Available in all 3.x series

# Built-in functions

- Iterator wrappers: `enumerate`, `zip`
- Boolean evaluators: `all`, `any`

See the python docs

# Iterator wrappers

Don't:

```
for i in range(len(a)):
    foo(i, a[i])
```

Do:

```
for i, v in enumerate(a):
    foo(i, v)
```

Available in all 3.x series

# Iterator wrappers

Don't:

```
for i in range(len(a)):
    foo(a[i], b[i])
```

Do:

```
for v, w in zip(a, b):
    foo(v, w)
```

# Boolean evaluators

```
for i in a:
    if i % 2:
        print("There is an odd number")
        break
```

```
for i in a:
    if not i % 2:
        break
else:
    print("They are all odd numbers")
```

# Boolean evaluators

```
if any(i % 2 for i in a):
    print("There is an odd number")
```

```
if all(i % 2 for i in a):
    print("They are all odd numbers")
```

Available in all 3.x series

# Built-in types: new *str* methods

```
>>> s = "This is a sample string"
>>> s.removeprefix("This")
' is a sample string'
>>> s.removesuffix("ing")
'This is a sample str'
```

Available since 3.9 (Oct 2021)

See the python docs

# Path manipulation

- `pathlib` — Object-oriented filesystem paths
- `shutil` — High-level file operations
- `tempfile` — Generate temporary files and directories

# "The" line

```
from pathlib import Path
```

Available since 3.4 (Mar 2014)

See the python docs

# Path object initialization

```
from pathlib import Path

# p points to any/path/you/want
p = Path("any", "path", "you", "want")
# Current Working Directory
cwd = Path.cwd()
# Home Directory; since 3.5 (Sep 2015)
cwd = Path.home()
# Home Directory of user jnooree; since 3.5
p = Path("~jnooree").expanduser()
```

Available since 3.4 (Mar 2014) (unless otherwise noted)

# Properties of Path object

```
>>> p = Path("any", "path", "you", "want")
>>> p
PosixPath('any/path/you/want')
>>> p.parts
('any', 'path', 'you', 'want')
>>> p.parent
PosixPath('any/path/you')
>>> p.parents[1]
PosixPath('any/path')
>>> p.parents[2:]  # Since 3.10 (Oct 2021)
(PosixPath('any'), PosixPath('.'))
```

Available since 3.4 (Mar 2014) (unless otherwise noted)

# Properties of Path object

```
>>> p = Path("foo", "bar.dir")
>>> p.name
bar.dir
>>> p.stem
bar
>>> p.suffix
.dir
```

Available since 3.4 (Mar 2014)

# Path manipulations

```
>>> p = Path("foo", "bar.dir")
>>> p / "baz.txt"
PosixPath('foo/bar.dir/baz.txt')
>>> p.joinpath("baz", "qux.txt")
PosixPath('foo/bar.dir/baz/qux.txt')
>>> p.with_name("quux")
PosixPath('foo/quux')
>>> p.with_stem("quz")  # Since 3.9 (Oct 2020)
PosixPath('foo/quz.dir')
>>> p.with_suffix(".d")
PosixPath('foo/bar.d')
```

Available since 3.4 (Mar 2014) (unless otherwise noted)

# Path manipulations

```
>>> p = Path("foo", "bar.dir")
>>> p.resolve()  # Make path absolute
PosixPath('/home/jnooree/foo/bar.dir')
>>> p.resolve(strict=True)  # Since 3.6 (Dec 2016)
FileNotFoundError: No such file or directory (...)
```

Available since 3.4 (Mar 2014) (unless otherwise noted)

# Querying metadata

```
p = Path("foo", "bar.txt")
# Query existence of path
p.exists()
# Query if the path exists and is a directory
p.is_dir()
# Query if the path exists and is a regular file
p.is_file()
# Query if the path is an absolute path
p.is_absolute()
```

Available since 3.4 (Mar 2014)

# Directory related operations

```
d = Path("foo", "bar")
# Equivalent to mkdir foo/bar
d.mkdir()
# Equivalent to mkdir -p foo/bar; since 3.5
d.mkdir(parents=True, exist_ok=True)
# Iterator of directory contents
for item in d.iterdir(): ...
# Iterator of directory contents matching pattern
for text in d.glob("*.txt"): ...
# Recursive globbing also available
for text in d.glob("**/*.txt"): ...
```

Available since 3.4 (Mar 2014) (unless otherwise noted)

# File related operations

```
text = Path("foo", "bar.txt")
# Equivalent to open(text)
with text.open() as f:
    for line in f:
        ...
# Whole content of the file; since 3.5 (Sep 2015)
content = text.read_text()
# Write to the file; since 3.5
text.write_text("This is a sample\n")
```

Available since 3.4 (Mar 2014) (unless otherwise noted)

# Moving and removing

```python
p = Path("foo", "bar")
# Rename foo/bar to baz (can't overwrite directory)
p.rename("baz")
# Force rename foo/bar to baz
p.replace("baz")
# For files:
# Equivalent to rm foo/bar
p.unlink()
# Equivalent to rm -f foo/bar; since 3.8 (Oct 2019)
p.unlink(missing_ok=True)
# For empty directories:
p.rmdir()
```

Available since 3.4 (Mar 2014) (unless otherwise noted)

# We all did this once

```
import os
os.system("rm -rf tmp")
```

# ... but it's **extremely** dangerous

```python
import os
#                          ↓ Oops!
p = "/home/jnooree /tmp"
# Will delete your whole home directory
os.system(f"rm -rf {p}")
```

# The rescue

```
import shutil
```

Available in all 3.x series

See the python docs

# Higher level file operations

```
import shutil

# Copy file data and permission
shutil.copy("src.txt", "dst.txt")
# Copy file data and metadata (modified time, ...)
shutil.copy2("src.txt", "dst.txt")
# cp -a src dst, except raises error if dst exists
shutil.copytree("src", "dst")
# rsync -a src/ dst/; since 3.8 (Oct 2019)
shutil.copytree("src", "dst", dirs_exist_ok=True)
# rm -r src
shutil.rmtree("src")
```

Available in all 3.x series (unless otherwise noted)

# Temporary files

Don't:

```
with open("temp.txt", "w") as f: ...
```

Do:

```
import tempfile as tmpf

with tmpf.TemporaryFile("w") as f: ...
with tmpf.NamedTemporaryFile("w") as f:
    foo(f.name)
    ...
```

Available in all 3.x series

# Temporary directories

Don't:

```
os.mkdir("temp")
...
os.system("rm -rf temp")
```

Do:

```
# The directory will be automatically deleted
with tmpf.TemporaryDirectory() as tmpdirname:
    ...
```

Available since 3.2 (Feb 2011)

# `collections` — Container datatypes

- `defaultdict`
- `Counter`

See the python docs

# A common workflow

```
l = [("yellow", 1), ("blue", 2), ("yellow", 3), ...]
d = {}
for color, value in l:
    if color not in d:
        d[color] = []
    d[color].append(value)
```

# The better workflow

```
from collections import defaultdict


l = [("yellow", 1), ("blue", 2), ("yellow", 3), ...]
d = defaultdict(list)
for color, value in l:
    d[color].append(value)
```

# Counting occurrences

```
l = ["foo", "bar", "bar", "baz", ...]
c = {}
for item in l:
    if item not in c:
        c[item] = 0
    c[item] += 1
```

# Oh, `defaultdict`, my `defaultdict`

```
from collections import defaultdict


l = ["foo", "bar", "bar", "baz", ...]
c = defaultdict(int)
for item in l:
    c[item] += 1
```

# Use the right tool

```
from collections import Counter

l = ["foo", "bar", "bar", "baz", ...]
c = Counter(l)
>>> c.most_common()
[('bar', 10), ('foo', 7), ('baz', 3), ...]
```

Available since 3.1 (Jun 2009)

# Everyone likes `os.system`

```
os.system("obrms ref.mol2 test.mol2")
```

# What if..?

```
os.system(f"obrms {ref} {test}")
```

# Never trust anyone

```
ref = "ref.mol2 test.mol2"
test = "; rm -rf /home/jnooree"

# Will calculate rmsd and delete your home directory
os.system(f"obrms {ref} {test}")
```

# Quoting might work?

```
ref = "ref.mol2 test.mol2"
test = "; rm -rf /home/jnooree"

# Will just throw error
os.system(f"obrms '{ref}' '{test}'")
```

# Not at all

```
ref = "ref.mol2 test.mol2"
test = "'; rm -rf /home/jnooree'"

# Will delete your home directory and throw error
os.system(f"obrms '{ref}' '{test}'")
```

## `subprocess` — Subprocess management

```
import subprocess as sp

sp.run(["obrms", ref, test])
```

Available since 3.5 (Sep 2015)

See the python docs

# Effortless output capturing

```
import subprocess as sp

ret = sp.run(["obrms", ref, test], stdout=sp.PIPE)
>>> ret.stdout
b'RMSD ref:test 53.7078\n'
```

Available since 3.5 (Sep 2015)

# Want string?

```
import subprocess as sp

ret = sp.run(["obrms", ref, test], stdout=sp.PIPE,
             text=True)
>>> ret.stdout
'RMSD ref:test 53.7078\n'
```

Available since 3.7 (Jun 2018)

# Need to check error?

```
>>> import subprocess as sp
>>> ret = sp.run(["rm", "/"])  # No error?
rm: cannot remove '/': Is a directory
```

Available since 3.5 (Sep 2015)

# Option 1: check return code

```
>>> import subprocess as sp
>>> ret = sp.run(["rm", "/"])  # No error?
rm: cannot remove '/': Is a directory
>>> ret.returncode  # 0 indicates successful run
1
```

Available since 3.5 (Sep 2015)

# Option 2: check

```
>>> import subprocess as sp
>>> ret = sp.run(["rm", "/"], check=True)
rm: cannot remove '/': Is a directory
Traceback (most recent call last):
  ...
subprocess.CalledProcessError: Command '['rm', '/']'
returned non-zero exit status 1.
```

Available since 3.5 (Sep 2015)

# Miscellaneous useful libraries

- `logging` — Logging facility for Python
- `itertools` — Functions for efficient looping
- `datetime` — Basic date and time types
- `argparse` — Parser for command-line
- `shlex` — Simple lexical analysis

# `logging` — Logging facility for Python

```
import logging

logging.basicConfig(level=logging.DEBUG)
# Prints 'DEBUG:root:This is a test' to stderr
logging.debug("This is %s", "a test")
# Set logging level to warning
logging.getLogger().setLevel(logging.WARNING)
# Does nothing
logging.debug("This is %s", "a test")
# Prints 'WARNING:root:This is another test'
logging.warning("This is %s", "another test")
```

Available in all 3.x series

See the python docs

## `itertools` — Functions for efficient looping

```
import itertools

for i, j in itertools.combinations(range(3), 2):
    # prints 0 1, 0 2, 1 2
    print(i, j)

for i, j in itertools.product(range(2), range(3)):
    # prints 0 0, 0 1, 0 2, 1 0, 1 1, 1 2
    print(i, j)
```

Available in all 3.x series

See the python docs

## `datetime` — Basic date and time types

```
>>> import datetime as dt
>>> now = dt.datetime.now()
>>> print(now)
'2022-07-07 10:47:20.913444'
>>> day_after = dt.timedelta(days=1)
>>> now + day_after
datetime.datetime(2022, 7, 8, 10, 47, 20, 913444)
>>> dt.date.today()
datetime.date(2022, 7, 7)
>>> dt.date.today() + day_after
datetime.date(2022, 7, 8)
```

Available in all 3.x series

See the python docs

# `argparse` — Parser for command-line

```
10  opt = Galaxy.core.ArgumentParser(
11      method='GalaxyDock2 with BP2 Score',
12      description='Protein-ligand docking with Pre-docking & BP2 Score')
13  #
14  opt.add_argument('-p', '--pdb', dest='pdb_fn', metavar='PDB', required=True,
15                   help='Input PDB to do ligand docking')
16  opt.add_argument('-l', '--lig', dest='lig_fn', metavar='LIG', required=True,
17                   help='Ligand structure file(multiple)')
18  opt.add_argument('-r', '--residue', dest='residue', metavar='POCKET',
19                   default=None, help='Known residues in binding pocket')
20  opt.add_argument('-c', '--center', metavar='CENTER', default=None,
21                   help=('Known residue center coordinate file, '
22                         'delimited by whitespaces'))
23  opt.add_argument('-prep', '--prep_method', dest='prep_method',
24                   metavar='PREP_METHOD', default='babel',
25                   help=('Input, cofactor preparation method '
26                         '(babel or chimera) [default: babel]'))
27  #
28  if len(sys.argv) == 1:
29      opt.print_help()
30      return
31  fn = opt.parse_args()
```

Available since 3.2 (Feb 2011)

Taken from the GalaxyPipe source code; See the python docs

# `shlex` — Simple lexical analysis

```
>>> import shlex
>>> cmd = "rm 'A file.txt'"
>>> shlex.split(cmd)
['rm', 'A file.txt']
>>> args = ["rm", "A file.txt"]
# Since 3.8 (Oct 2019)
>>> shlex.join(args)
"rm 'A file.txt'"
# Since 3.3 (Sep 2012)
>>> print(shlex.quote("I'm foo.txt"))
'I'"'"'m foo.txt'
```

Available in all 3.x series (unless otherwise noted)

See the python docs

# Wait, what about X?

- `glob` — Use `pathlib`
- `math`, `random`, `statistics` — Use `numpy` & `scipy`
- `csv` — Use `pandas`
- `multiprocessing` — Use `joblib`
- `threading` — Not suitable for parallelizing CPU-bound tasks
- `os` — Pile of low-level historic functions
- `sys` — Not very interesting
- `time` — Replaceable with `datetime`
  (Exception: `time.sleep`, `time.perf_counter`)
- `re` — Regex is another whole lecture
- `pickle` — Everyone knows how to pickle
- `functools` — Why did you attend this seminar?

# Part V: Lab Session

Now it's your turn!

```python
# Extract all sequence and chemical component details, and
# populate seq_to_structure_mappings using author chain IDs.
author_chain_to_sequence = defaultdict(str)
all_chem_comp_details = defaultdict(list)
seq_to_structure_mappings = defaultdict(dict)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]
    current_mapping = seq_to_structure_mappings[author_chain]
    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
    seq = []
    all_chem_comp_info = []
    for monomer_index, monomer in enumerate(seq_info):
        all_chem_comp_info.append(chem_comp_info[monomer_index])
        if "peptide" in chem_comp_info[monomer_index].type.lower():
            code = PDBData.protein_letters_3to1.get(f"{monomer.id: <3}", "X")
        elif (
            "dna" in chem_comp_info[monomer_index].type.lower()
            or "rna" in chem_comp_info[monomer_index].type.lower()
        ):
            code = PDBData.nucleic_letters_3to1.get(f"{monomer.id: <3}", "X")
        else:
            code = "X"
        seq.append(code if len(code) == 1 else "X")
    author_chain_to_sequence[author_chain] += "".join(seq)
    all_chem_comp_details[author_chain].extend(all_chem_comp_info)
    if current_mapping:
        start_index = len(current_mapping)
        current_mapping.update(
            {
                start_index + value_index: value
                for value_index, value in enumerate(mmcif_current_mapping.values())
            }
        )
    else:
        current_mapping.update(mmcif_current_mapping)
```

Taken from alphafold3-pytorch source code

```python
# Extract all sequence and chemical component details, and
# populate seq_to_structure_mappings using author chain IDs.
author_chain_to_sequence = defaultdict(str)
all_chem_comp_details = defaultdict(list)
seq_to_structure_mappings = defaultdict(dict)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]
    current_mapping = seq_to_structure_mappings[author_chain]
    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
    seq = []
    all_chem_comp_info = []
    for monomer_index, monomer in enumerate(seq_info):
        all_chem_comp_info.append(chem_comp_info[monomer_index])
        if "peptide" in chem_comp_info[monomer_index].type.lower():
            code = PDBData.protein_letters_3to1.get(f"{monomer.id: <3}", "X")
        elif (
            "dna" in chem_comp_info[monomer_index].type.lower()
            or "rna" in chem_comp_info[monomer_index].type.lower()
        ):
            code = PDBData.nucleic_letters_3to1.get(f"{monomer.id: <3}", "X")
        else:
            code = "X"
        seq.append(code if len(code) == 1 else "X")
    author_chain_to_sequence[author_chain] += "".join(seq)
    all_chem_comp_details[author_chain].extend(all_chem_comp_info)
    if current_mapping:
        start_index = len(current_mapping)
        current_mapping.update(
            {
                start_index + value_index: value
                for value_index, value in enumerate(mmcif_current_mapping.values())
            }
        )
    else:
        current_mapping.update(mmcif_current_mapping)
```

# Nope

Taken from alphafold3-pytorch source code

```
    author_chain_to_sequence = defaultdict(str)
    all_chem_comp_details = defaultdict(list)
    seq_to_structure_mappings = defaultdict(dict)
    for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
        author_chain = mmcif_to_author_chain_id[chain_id]
        current_mapping = seq_to_structure_mappings[author_chain]
        mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
        seq = []
        all_chem_comp_info = []
        for monomer_index, monomer in enumerate(seq_info):
            all_chem_comp_info.append(chem_comp_info[monomer_index])
            if "peptide" in chem_comp_info[monomer_index].type.lower():
                code = PDBData.protein_letters_3to1.get(f"{monomer.id: <3}", "X")
            elif (
                "dna" in chem_comp_info[monomer_index].type.lower()
                or "rna" in chem_comp_info[monomer_index].type.lower()
            ):
                code = PDBData.nucleic_letters_3to1.get(f"{monomer.id: <3}", "X")
            else:
                code = "X"
            seq.append(code if len(code) == 1 else "X")
        author_chain_to_sequence[author_chain] += "".join(seq)
        all_chem_comp_details[author_chain].extend(all_chem_comp_info)
        if current_mapping:
            start_index = len(current_mapping)
            current_mapping.update(
                {
                    start_index + value_index: value
                    for value_index, value in enumerate(mmcif_current_mapping.values())
                }
            )
        else:
            current_mapping.update(mmcif_current_mapping)
```

Taken from alphafold3-pytorch source code

```python
author_chain_to_sequence = defaultdict(str)
all_chem_comp_details = defaultdict(list)
seq_to_structure_mappings = defaultdict(dict)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]
    current_mapping = seq_to_structure_mappings[author_chain]
    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
    seq = []
    all_chem_comp_info = []
    for monomer_index, monomer in enumerate(seq_info):
        all_chem_comp_info.append(chem_comp_info[monomer_index])
        if "peptide" in chem_comp_info[monomer_index].type.lower():
            code = PDBData.protein_letters_3to1.get(f"{monomer.id: <3}", "X")
        elif (
            "dna" in chem_comp_info[monomer_index].type.lower()
            or "rna" in chem_comp_info[monomer_index].type.lower()
        ):
            code = PDBData.nucleic_letters_3to1.get(f"{monomer.id: <3}", "X")
        else:
            code = "X"
        seq.append(code if len(code) == 1 else "X")
    author_chain_to_sequence[author_chain] += "".join(seq)
    all_chem_comp_details[author_chain].extend(all_chem_comp_info)
    if current_mapping:
        start_index = len(current_mapping)
        current_mapping.update(
            {
                start_index + value_index: value
                for value_index, value in enumerate(mmcif_current_mapping.values())
            }
        )
    else:
        current_mapping.update(mmcif_current_mapping)
```

# To single letter

Taken from alphafold3-pytorch source code

```
author_chain_to_sequence = defaultdict(str)
all_chem_comp_details = defaultdict(list)
seq_to_structure_mappings = defaultdict(dict)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]
    current_mapping = seq_to_structure_mappings[author_chain]
    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]

    author_chain_to_sequence[author_chain] += seq_info
    all_chem_comp_details[author_chain].extend(chem_comp_info)

    if current_mapping:
        start_index = len(current_mapping)
        current_mapping.update(
            {
                start_index + value_index: value
                for value_index, value in enumerate(mmcif_current_mapping.values())
            }
        )
    else:
        current_mapping.update(mmcif_current_mapping)
```

Taken from alphafold3-pytorch source code

```python
author_chain_to_sequence = defaultdict(str)
all_chem_comp_details = defaultdict(list)
seq_to_structure_mappings = defaultdict(dict)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]
    current_mapping = seq_to_structure_mappings[author_chain]
    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]

    author_chain_to_sequence[author_chain] += seq_info
    all_chem_comp_details[author_chain].extend(chem_comp_info)

    if current_mapping:
        start_index = len(current_mapping)
        current_mapping.update(
            {
                start_index + value_index: value
                for value_index, value in enumerate(mmcif_current_mapping.values())
            }
        )
    else:
        current_mapping.update(mmcif_current_mapping)
```

# Why not list?

Taken from alphafold3-pytorch source code

```
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(list)
480 for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
481     author_chain = mmcif_to_author_chain_id[chain_id]
482     current_mapping = seq_to_structure_mappings[author_chain]
483     mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
484 
485     author_chain_to_sequence[author_chain] += seq_info
486     all_chem_comp_details[author_chain].extend(chem_comp_info)
487 
488     current_mapping.extend(mmcif_current_mapping.values())
```

```
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(list)
480 for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
481     author_chain = mmcif_to_author_chain_id[chain_id]
482     current_mapping = seq_to_structure_mappings[author_chain]
483     mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
484
485     author_chain_to_sequence[author_chain] += seq_info
486     all_chem_comp_details[author_chain].extend(chem_comp_info)
487
488     current_mapping.extend(mmcif_current_mapping.values())
```

# Name them properly

```
477     chain_to_sequence = defaultdict(str)
478     chain_to_chem_comp = defaultdict(list)
479     chain_to_structure = defaultdict(list)
480     for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
481         author_chain = mmcif_to_author_chain_id[chain_id]
482         current_mapping = chain_to_structure[author_chain]
483         mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
484 
485         chain_to_sequence[author_chain] += seq_info
486         chain_to_chem_comp[author_chain].extend(chem_comp_info)
487 
488         current_mapping.extend(mmcif_current_mapping.values())
```

```
chain_to_sequence = defaultdict(str)
chain_to_chem_comp = defaultdict(list)
chain_to_structure = defaultdict(list)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]
    current_mapping = chain_to_structure[author_chain]
    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]

    chain_to_sequence[author_chain] += seq_info
    chain_to_chem_comp[author_chain].extend(chem_comp_info)

    current_mapping.extend(mmcif_current_mapping.values())
```

```
chain_to_sequence = defaultdict(str)
chain_to_chem_comp = defaultdict(list)
chain_to_structure = defaultdict(list)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]

    chain_to_sequence[author_chain] += seq_info
    chain_to_chem_comp[author_chain].extend(chem_comp_info)

    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
    chain_to_structure[author_chain].extend(mmcif_current_mapping.values())
```

```
chain_to_sequence = defaultdict(str)
chain_to_chem_comp = defaultdict(list)
chain_to_structure = defaultdict(list)
for chain_id, (seq_info, chem_comp_info) in valid_chains.items():
    author_chain = mmcif_to_author_chain_id[chain_id]

    chain_to_sequence[author_chain] += seq_info
    chain_to_chem_comp[author_chain].extend(chem_comp_info)

    mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
    chain_to_structure[author_chain].extend(mmcif_current_mapping.values())

```

# Why track separately?

```
325  class Chain:
326      ...
```

```
477  chains: dict[str, Chain] = valid_chains
```

```
325  class Chain:
326      ...
```

# Appendix: PDB file format

- Fixed-width (column X is A, Y is B, ...)

```
import pdbline
```

- Each line starts with a *record identifier* (`record`)
  - You will need `ATOM` and `HETATM` records for atomic coordinates
  - `MODEL` records will be useful for later tasks
- A *residue* is identified by a combination of
  - Chain ID (`chain_id`),
  - Residue sequence number (`res_seq`),
  - And insertion code (`ins_code`)
- To identify an *atom*, need atom name (`atom_name`) additionally

*PDB format specification link

(numpy & stdlib allowed)

# Code, code, code

PDB 파일이 주어졌을 때, 모든 원자의 center of mass coordinate?

*PDB format specification link

# Where to go next

- CodeAesthetic YouTube channel
- Kevlin Henney - Giving code a good name
- Philomatics Git cheatsheet
- Wilson, G. et al. Best Practices for Scientific Computing. *PLOS Biology* **2014**, *12* (1), e1001745. DOI
- Refactoring Guru website
- Brandon Rhodes blog
- Martin Fowler blog

ChatGPT helped pick a few - quality may vary.™

# Q&A

*Thank you for Listening*

# Appendix: SOLID principles of OOP

- The **S**ingle-responsibility principle
- The **O**pen-closed principle
- The **L**iskov substitution principle
- The **I**nterface segregation principle
- The **D**ependency inversion principle

# Single-responsibility principle

`torch.utils.data.Dataset`

`torch.utils.data.DataLoader`

# Single-responsibility principle

```
class MyDataset(Dataset):
    def __init__(self, path):
        self.path = path
        ...

dataset = MyDataset(path)
loader = DataLoader(dataset, ...)
```

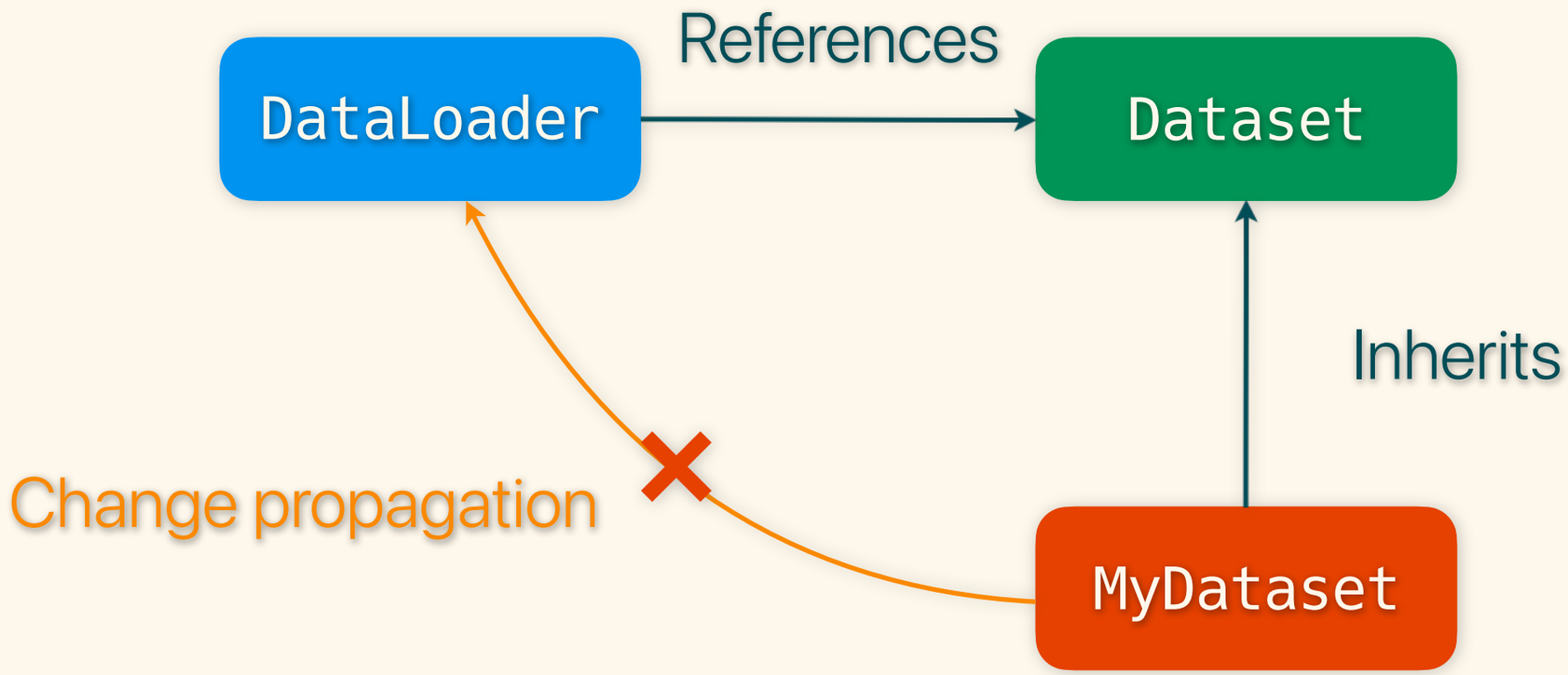
# Single-responsibility principle

```
class MyDataset(Dataset):
    def __init__(self, path, tmpdir):
        self.path = path
        self.tmpdir = tmpdir
        ...

dataset = MyDataset(path, tmpdir)
loader = DataLoader(dataset, ...)
```

# Dependency inversion principle

# Dependency inversion principle



# Dependency inversion principle

```
class MyDataset(Dataset):
    def __init__(self, path, tmpdir):
        self.path = path
        self.tmpdir = tmpdir
        ...

dataset = MyDataset(path, tmpdir)
loader = DataLoader(dataset, ...)
```

# Dependency inversion principle

```
class YourDataset(Dataset):
    def __init__(self, data):
        self.data = data
    ...

dataset = YourDataset(data)
loader = DataLoader(dataset, ...)
```

# Open-closed principle

```
class RNN:
    def run_rnn(self, ...): ...

class CNN:
    def run_cnn(self, ...): ...

if type(model) is RNN:
    h = model.run_rnn(...)
elif type(model) is CNN:
    h = model.run_cnn(...)
...
```

# Open-closed principle

```
class Module:
    def __call__(self, ...):
        self.forward(...)
# Open to extension
class RNN(Module):
    def forward(self, ...): ...

class CNN(Module):
    def forward(self, ...): ...

h = model(...)  # Closed for modification
```

# Liskov substitution principle

```python
class Predictor(nn.Module):
    def predict(self, x):
        self.eval()
        with torch.no_grad():
            return self(x)
```

```python
class DropoutPredictor(Predictor):
    def __init__(self, dim, dropout=0.5):
        super().__init__()
        self.model = nn.Linear(dim, dim)
        self.dropout = dropout

    def forward(self, x):
        return F.dropout(self.model(x), p=self.dropout)  # oops!
```

# Liskov substitution principle

```python
class Predictor(nn.Module):
    def predict(self, x):
        self.eval()
        with torch.no_grad():
            return self(x)
```

```python
class DropoutPredictor(Predictor):
    def __init__(self, dim, dropout=0.5):
        super().__init__()
        self.model = nn.Linear(dim, dim)
        self.dropout = dropout

    def forward(self, x):
        return F.dropout(self.model(x), p=self.dropout, training=self.training)
```

# Interface segregation principle

```
class Animal:
    def run(self, dest): ...

class Human(Animal):
    def run(self, dest): ...

class Salmon(Animal):
    def run(self, dest):
        raise ValueError("Salmon can't run")
```

# Interface segregation principle

```
class Animal: ...

class Runnable:
    def run(self, dest): ...

class Human(Runnable, Animal):
    def run(self, dest): ...

class Salmon(Animal): ...
```