
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 *unreadable*?

What makes code *unreadable*?

```
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 *unreadable*?

```
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 *unreadable*?

b = a(...)

e = b.c(d=1)

Coding is communication

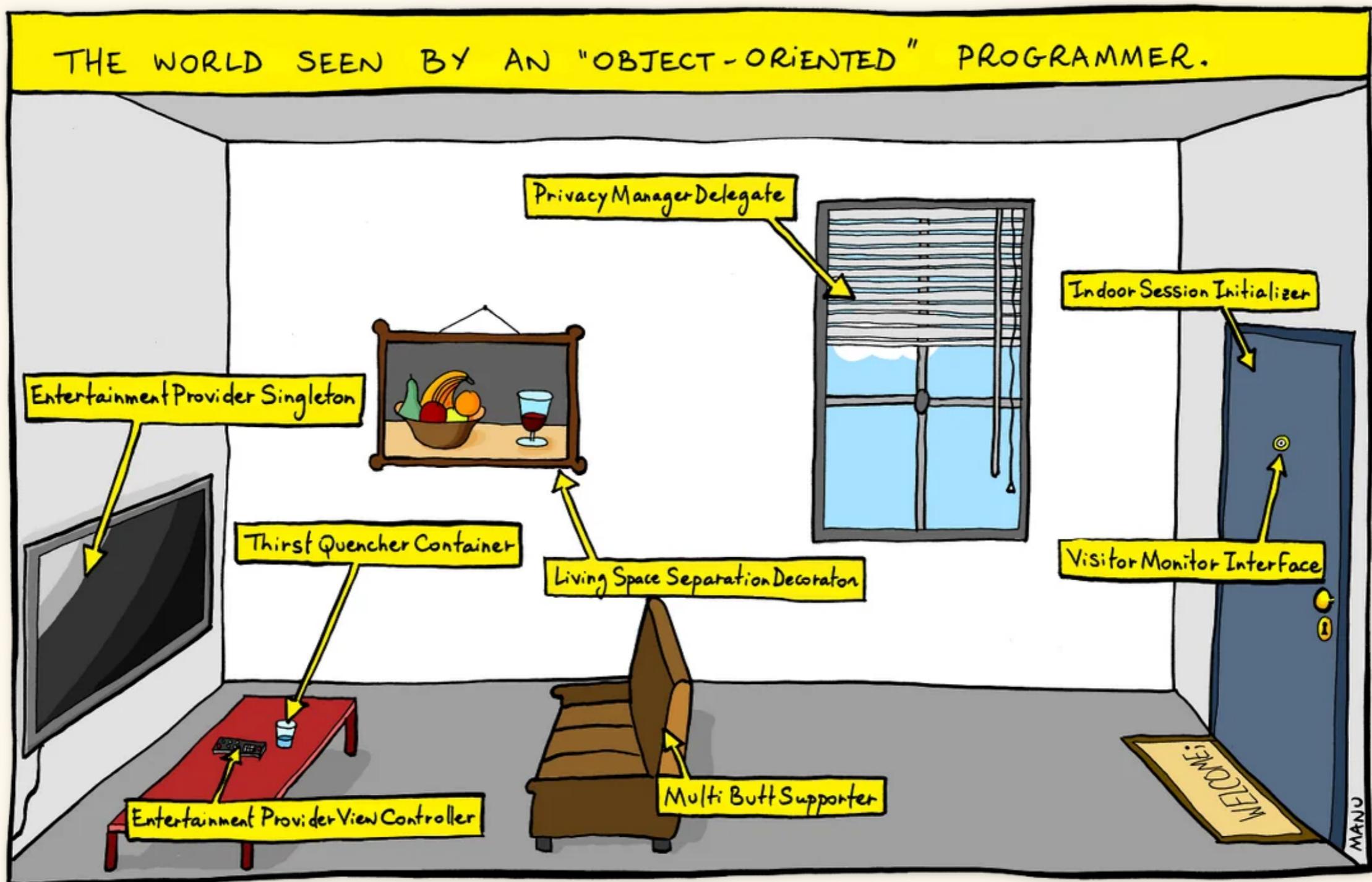
```
b = a(...)
```

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

```
arr = np.array(...)
```

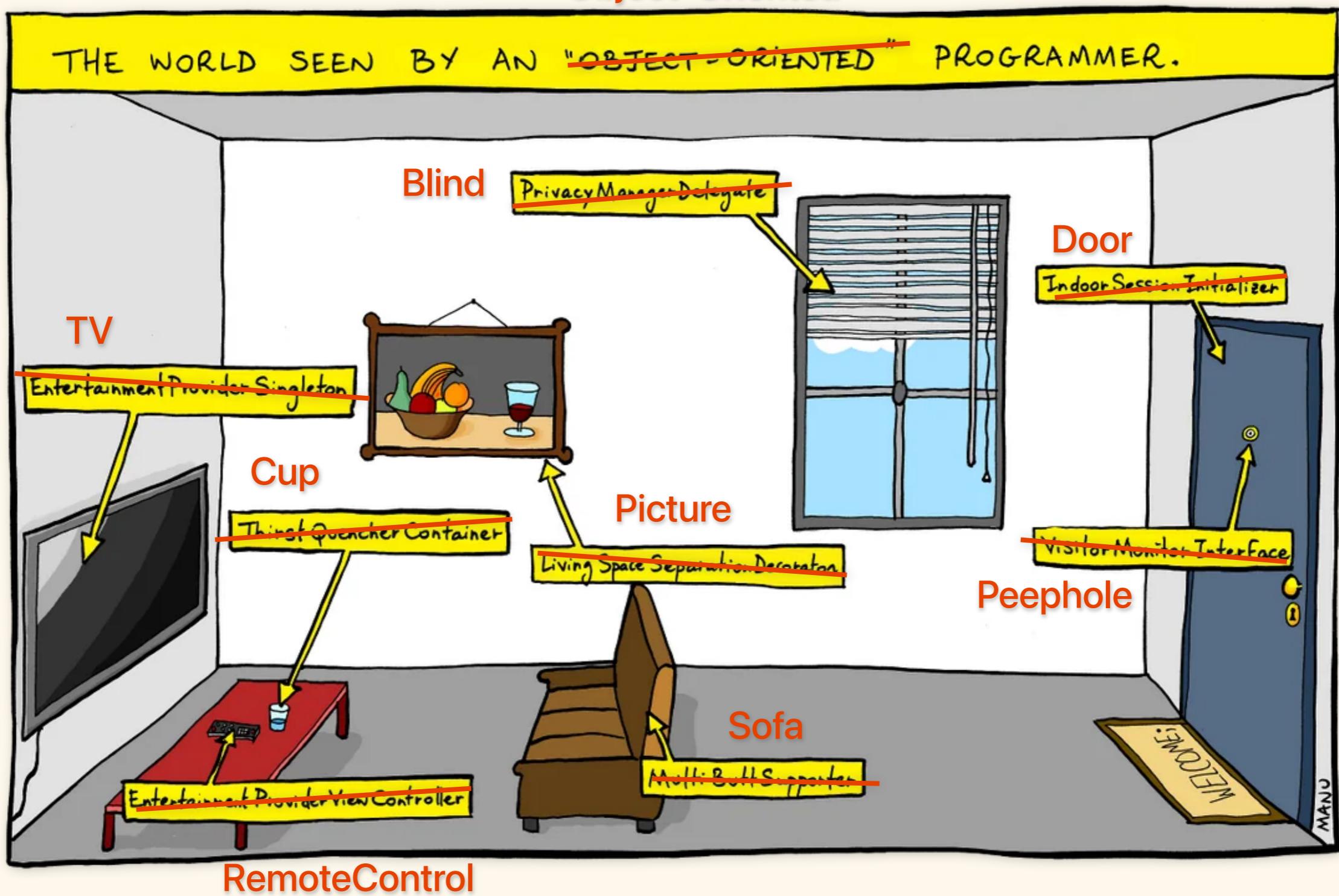
```
cntr = arr.mean(axis=1)
```

Don't be too noisy



Don't be too noisy

Object-Oriented



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 = ...
```

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)
```

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))
```

Because comments lie

```
164 # TODO: Reconstruct 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
```

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?

```
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

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

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?

```
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
```

```
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
```

```
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?

```
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
```

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

```
    ...
    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
```

```
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,
):
    s, len_seq = read_fasta(fa_fn)
    b = open(pdb_fn).readlines()

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

    s_idx = -1
    s_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?

```
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]  
        if atm_name == "CA":  
            ... # special handling for CA atom  
        tensor_coord[...] = torch.tensor(...)  
  
    return tensor_coord
```

```
def str_to_tensor_whole(  
    pdb_name,  
    use_all_atom,  
):  
    tensor_coord = torch.zeros(len_seq, len(target_atom), 3  
  
    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  
    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(...)  
  
        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
```

```
def str_to_tensor_wo_reference(  
    pdb_name,  
    pdb_fn: str,  
    fa_fn: str,  
    'lr_fn: str,  
    decoy_pdb_fn: str,  
    decoy_list: list,  
    target_atom,  
    decoy_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  
    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('lr_fn') == 3:  
            'lr_chain, from_res_no, to_res_no = 'lr_fn  
        else:  
            from_res_no, to_res_no, 'lr_chain = read_lr('lr_fn)  
            #  
            dic = ...  
    return dic
```

Extract *real* functions

```
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

The diagram illustrates the concept of keeping a single level of abstraction through three distinct sections of code:

- Mid-level:** A pink bracket highlights the line `fa_s, len_seq = read_fasta(fa_fn)`. A pink arrow points from this bracket to the first section of code.
- Low-level:** A blue bracket encloses the entire block of code starting with `pdb = open(pdb_fn).readlines()`. A blue arrow points from this bracket to the second section of code.
- High-level:** An orange bracket highlights the call `test2, decoy_flag = str_to_tensor_decoys(...)`. An orange arrow points from this bracket to the third section of code.

```
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
```

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

Don't hard-code

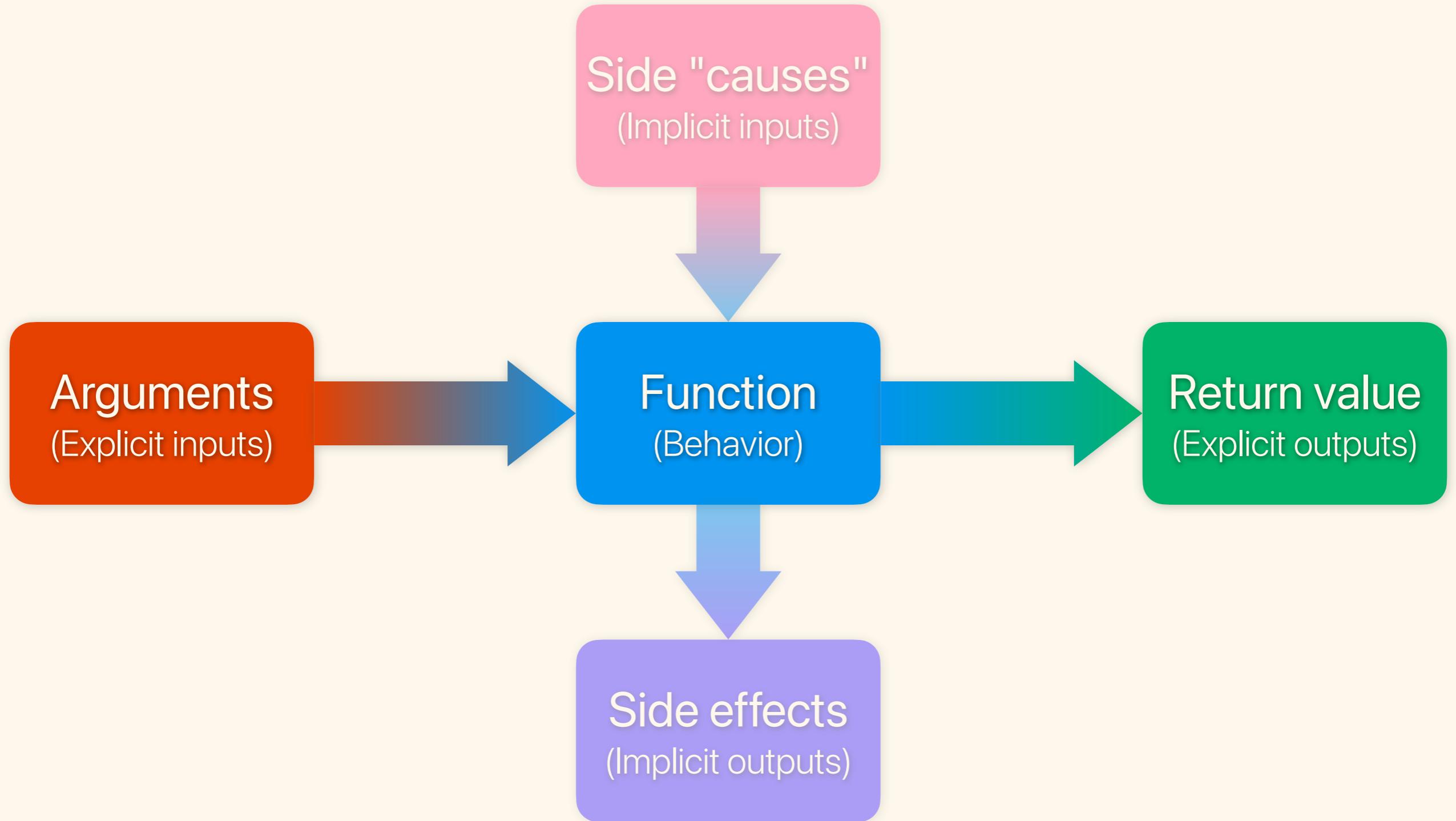
```
def build_graph(pdb_tag, decoy_type_num, ...):
    num_decoy_group = ...
    group_idx = ...
    graph_s = []
    ... # complex code filling graph_s
    if decoy_type_num:
        path = f"{GRAPH_PATH}/{pdb_tag}_{decoy_type_num}"
    else:
        path = f"{GRAPH_PATH}/{pdb_tag}"

    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)
```

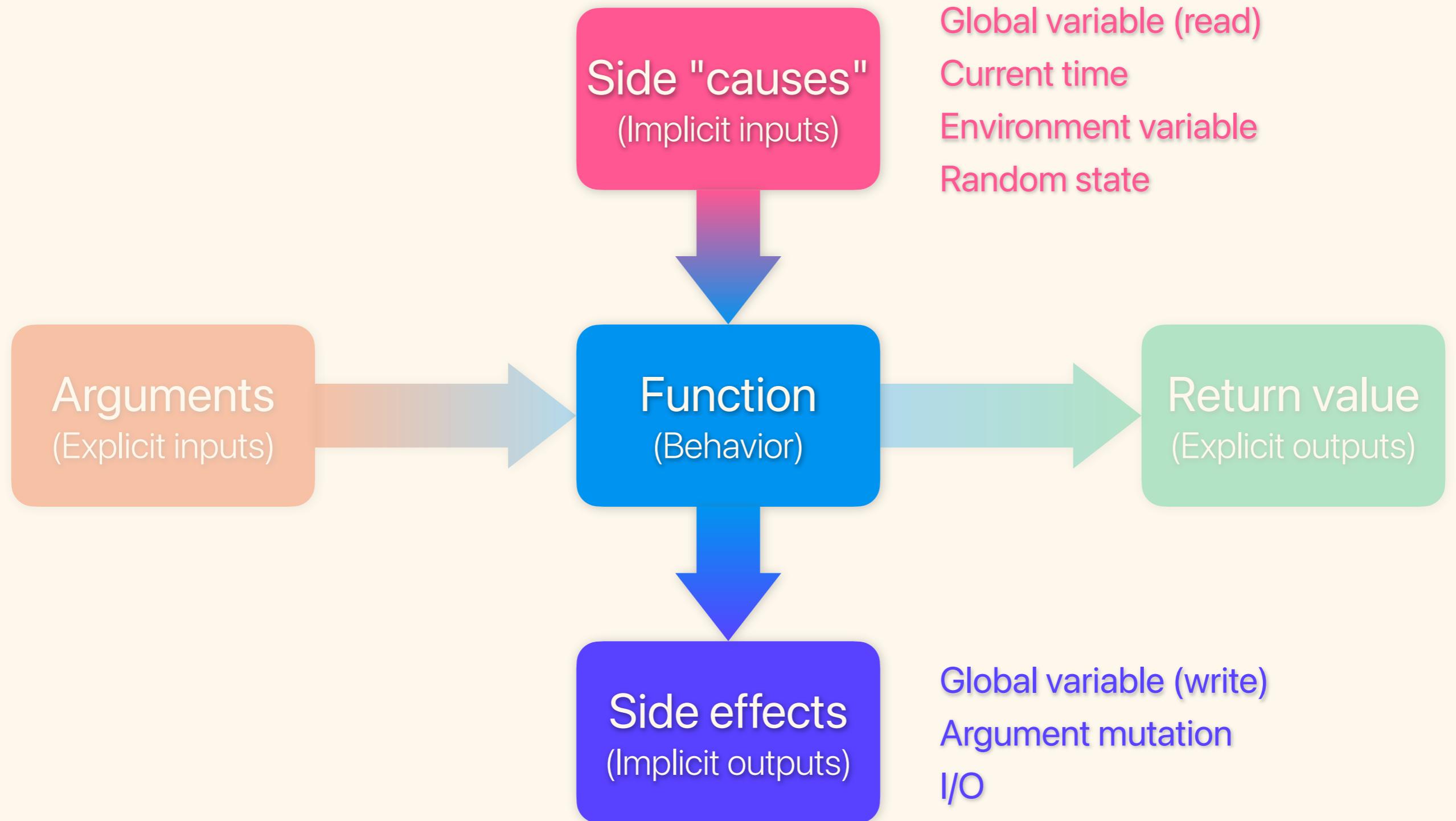
Must copy-paste this to use output

- 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?

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

```
[/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     ...
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1090
1091
1092
1093
1094
1095
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1485
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1515
1516
1517
1518
1518
1519
1520
1521
1522
1523
1524
1524
1525
1526
1527
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1544
1545
1546
1547
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1564
1565
1566
1567
1567
1568
1569
1569
1570
1571
1572
1573
1574
1574
1575
1576
1577
1577
1578
1579
1579
1580
1581
1582
1583
1584
1584
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1604
1605
1606
1607
1607
1608
1609
1609
1610
1611
1612
1613
1614
1614
1615
1616
1617
1617
1618
1619
1619
1620
1621
1622
1623
1624
1624
1625
1626
1627
1627
1628
1629
1629
1630
1631
1632
1633
1634
1634
1635
1636
1637
1637
1638
1639
1639
1640
1641
1642
1643
1644
1644
1645
1646
1647
1647
1648
1649
1649
1650
1651
1652
1653
1654
1654
1655
1656
1657
1657
1658
1659
1659
1660
1661
1662
1663
1664
1664
1665
1666
1667
1667
1668
1669
1669
1670
1671
1672
1673
1674
1674
1675
1676
1677
1677
1678
1679
1679
1680
1681
1682
1683
1684
1684
1685
1686
1687
1687
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1704
1705
1706
1707
1707
1708
1709
1709
1710
1711
1712
1713
1714
1714
1715
1716
1717
1717
1718
1719
1719
1720
1721
1722
1723
1724
1724
1725
1726
1727
1727
1728
1729
1729
1730
1731
1732
1733
1734
1734
1735
1736
1737
1737
1738
1739
1739
1740
1741
1742
1743
1744
1744
1745
1746
1747
1747
1748
1749
1749
1750
1751
1752
1753
1754
1754
1755
1756
1757
1757
1758
1759
1759
1760
1761
1762
1763
1764
1764
1765
1766
1767
1767
1768
1769
1769
1770
1771
1772
1773
1774
1774
1775
1776
1777
1777
1778
1779
1779
1780
1781
1782
1783
1784
1784
1785
1786
1787
1787
1788
1789
1789
1790
1791
1792
1793
1794
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1804
1805
1806
1807
1807
1808
1809
1809
1810
1811
1812
1813
1814
1814
1815
1816
1817
1817
1818
1819
1819
1820
1821
1822
1823
1824
1824
1825
1826
1827
1827
1828
1829
1829
1830
1831
1832
1833
1834
1834
1835
1836
1837
1837
1838
1839
1839
1840
1841
1842
1843
1844
1844
1845
1846
1847
1847
1848
1849
1849
1850
1851
1852
1853
1854
1854
1855
1856
1857
1857
1858
1859
1859
1860
1861
1862
1863
1864
1864
1865
1866
1867
1867
1868
1869
1869
1870
1871
1872
1873
1874
1874
1875
1876
1877
1877
1878
1879
1879
1880
1881
1882
1883
1884
1884
1885
1886
1887
1887
1888
1889
1889
1890
1891
1892
1893
1894
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1904
1905
1906
1907
1907
1908
1909
1909
1910
1911
1912
1913
1914
1914
1915
1916
1917
1917
1918
1919
1919
1920
1921
1922
1923
1924
1924
1925
1926
1927
1927
1928
1929
1929
1930
1931
1932
1933
1934
1934
1935
1936
1937
1937
1938
1939
1939
1940
1941
1942
1943
1944
1944
1945
1946
1947
1947
1948
1949
1949
1950
1951
1952
1953
1954
1954
1955
1956
1957
1957
1958
1959
1959
1960
1961
1962
1963
1964
1964
1965
1966
1967
1967
1968
1969
1969
1970
1971
1972
1973
1974
1974
1975
1976
1977
1977
1978
1979
1979
1980
1981
1982
1983
1984
1984
1985
1986
1987
1987
1988
1989
1989
1990
1991
1992
1993
1994
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2004
2005
2006
2007
2007
2008
2009
2009
2010
2011
2012
2013
2014
2014
2015
2016
2017
2017
2018
2019
2019
2020
2021
2022
2023
2024
2024
2025
2026
2027
2027
2028
2029
2029
2030
2031
2032
2033
2034
2034
2035
2036
2037
2037
2038
2039
2039
2040
2041
2042
2043
2044
2044
2045
2046
2047
2047
2048
2049
2049
2050
2051
2052
2053
2054
2054
2055
2056
2057
2057
2058
2059
2059
2060
2061
2062
2063
2064
2064
2065
2066
2067
2067
2068
2069
2069
2070
2071
2072
2073
2074
2074
2075
2076
2077
2077
2078
2079
2079
208
```

Effect of a dedicated return type

```
649 def nurikit_get_infos(  
650     id1: str,  
651 ) -> tuple[dict[str, list[str]], dict[str, dict[str, np.ndarray]], dict[str, list[str]]]:  
652     ...  
  
225 def nurikit_get_infos(  
226     pdb_id: str, pdb_orig: Path, tmpd: Path  
227 ) -> dict[str, ChainInfo]:  
228     ...  
  
41 class ChainInfo:  
42     seq_pieces: list[str]  
43     resnum_atom: list[str]  
44     coords: BackboneCoords  
34 class BackboneCoords:  
35     N: list[np.ndarray]  
36     CA: list[np.ndarray]  
37     C: list[np.ndarray]
```

Effect of a dedicated input type

```
714 def match_number(seq_res: dict[str, str], seq_pieces: dict[str, list[str]],  
715             resnum_atom: dict[str, list[str]], m_c: str):  
716     ...  
  
308 def match_number(seqres: str, chain_info: ChainInfo):  
309     ...  
  
34 class BackboneCoords:  
35     N: list[np.ndarray]  
36     CA: list[np.ndarray]  
37     C: list[np.ndarray]  
41 class ChainInfo:  
42     seq_pieces: list[str]  
43     resnum_atom: list[str]  
44     coords: BackboneCoords
```

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!

```
1 class Data:  
2     def __init__(self, a: int, b: str, c: bool):  
3         self.a = a  
4         self.b = b  
5         self.c = c
```

```
1 from dataclasses import dataclass  
2  
3 @dataclass  
4 class Data:  
5     a: int  
6     b: str  
7     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.

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:

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

Do:

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

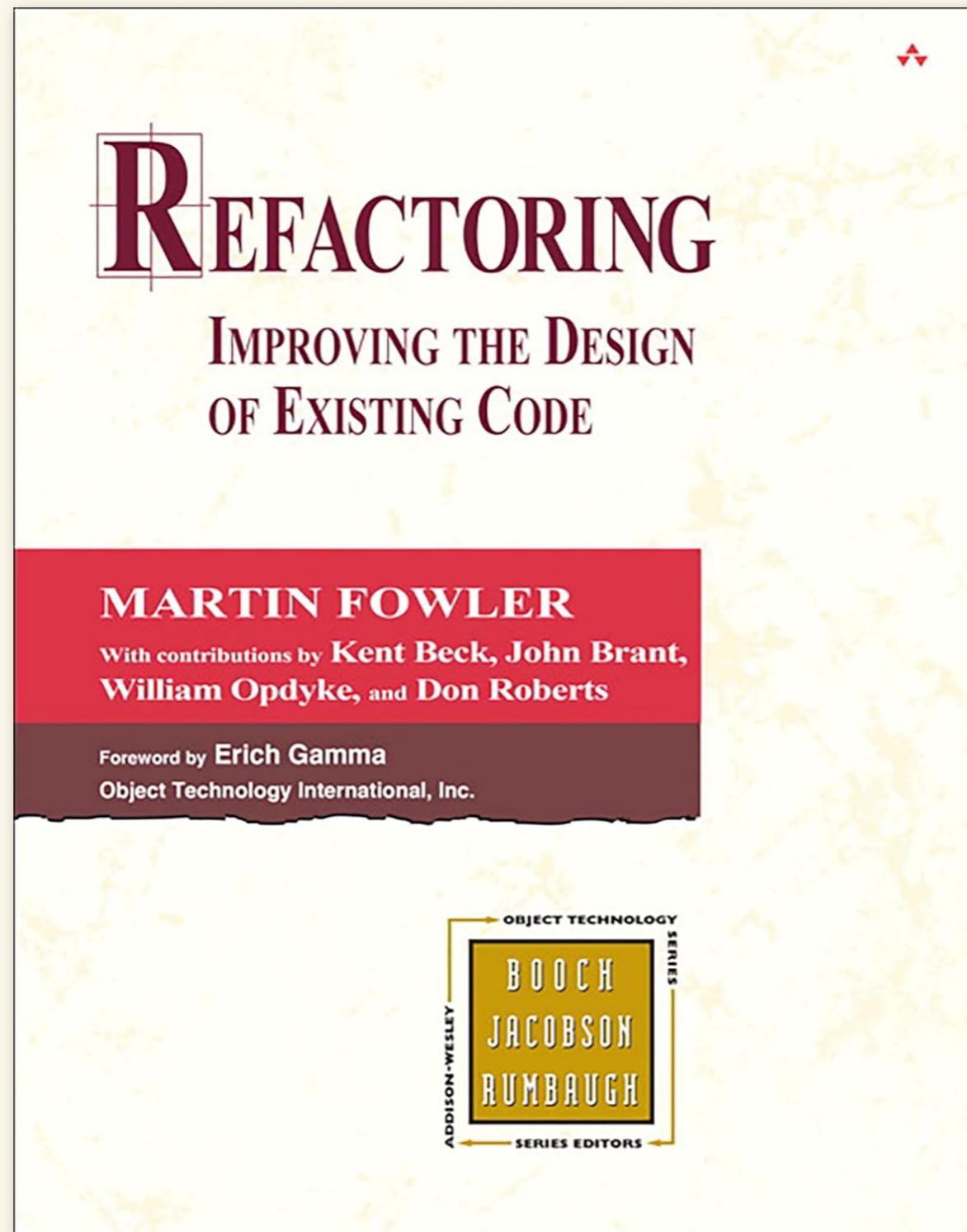
Tuple is for:

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

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)

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?

We need an *interface*

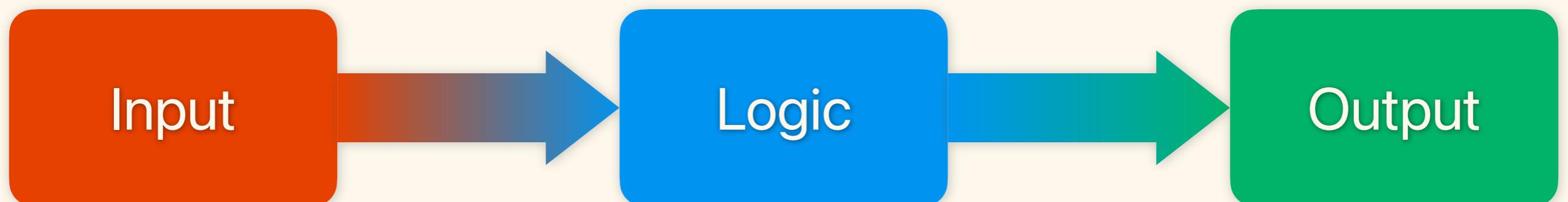
```
18 if isinstance(pdb, Galaxy.core.PDB):  
19     opt_s['infile_pdb'] = pdb.pdb_fn.relpath()  
20 elif isinstance(pdb, Galaxy.core.FilePath):  
21     opt_s['infile_pdb'] = pdb.relpath()  
22 elif isinstance(pdb, str):  
23     pdb_fn = Galaxy.core.FilePath(pdb)  
24     opt_s['infile_pdb'] = pdb_fn.relpath()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

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

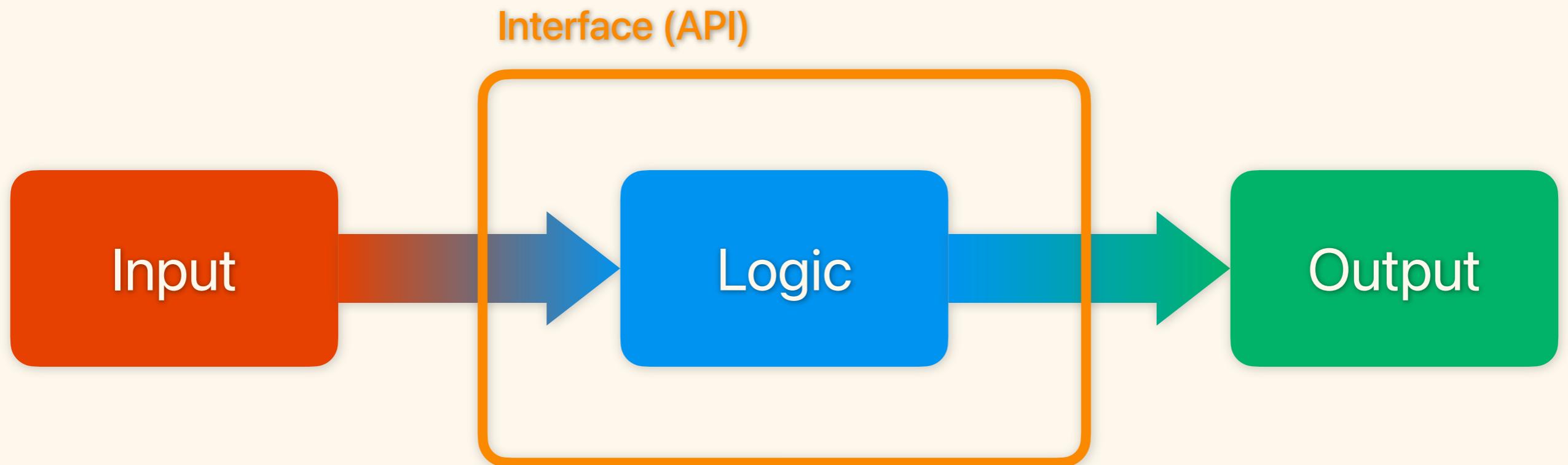
The **principles** behind this example are:

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

Any code has input and output



Interface is abstraction exposed



Object-oriented programming (\approx class)

The Feynman Problem Solving Algorithm:

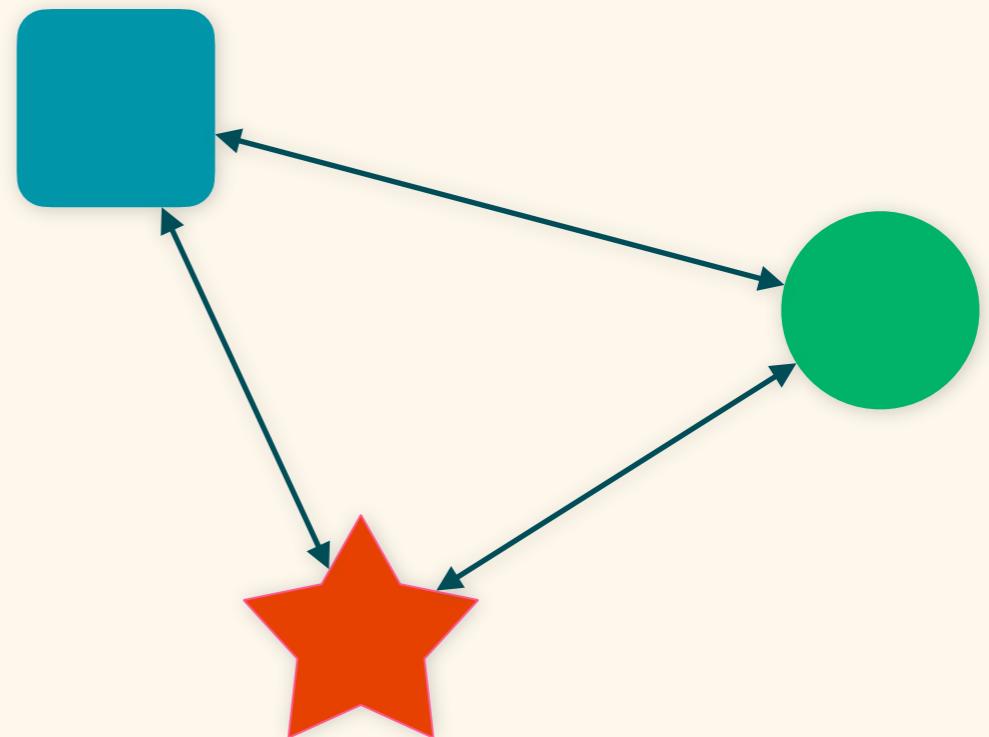
1. Write down the problem.

I
V

2. Think real hard.

I
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

```
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"

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

Avoid logic in initializer

```
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"]]
```

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

```
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"]],  
        )
```

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

Use factories

Avoid logic in initializer

```
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

Avoid logic in initializer

```
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
```

```
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)

Prefer functions over methods

```
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

```
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
```

Recap: FTSE

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

- Wheeler, D. J., FRS (emphasis mine)

Recap: FTSE + a

*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?

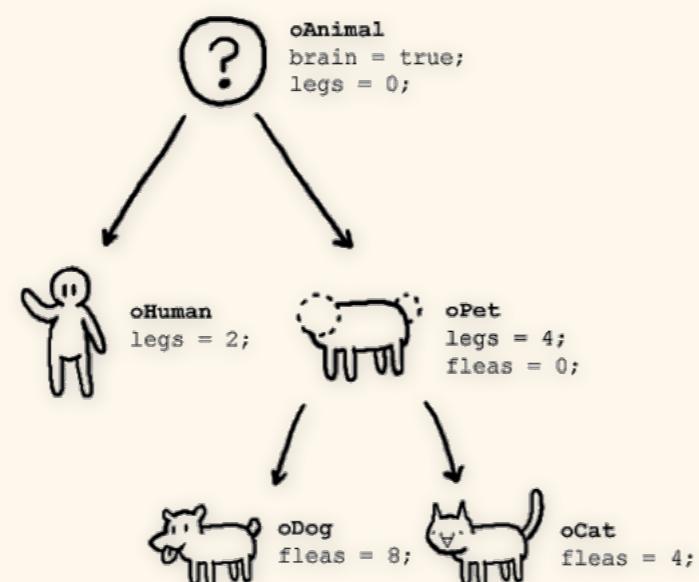
- Readable
- Working
- Simple
- Maintainable
- Extensible & reusable
- Self-documenting

Recap: what is good code?

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

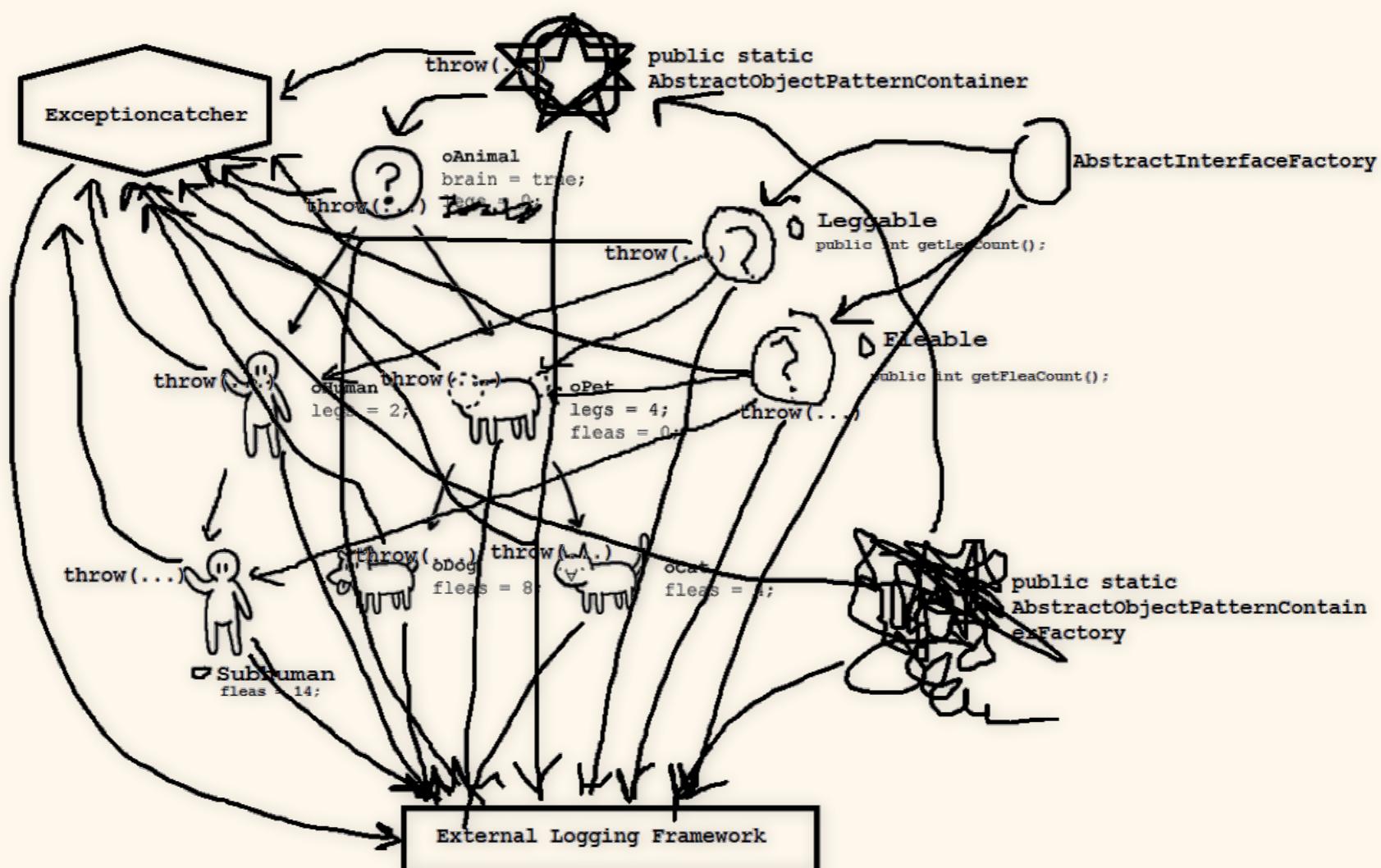
OOP in real world

What should have been done



OOP in real world

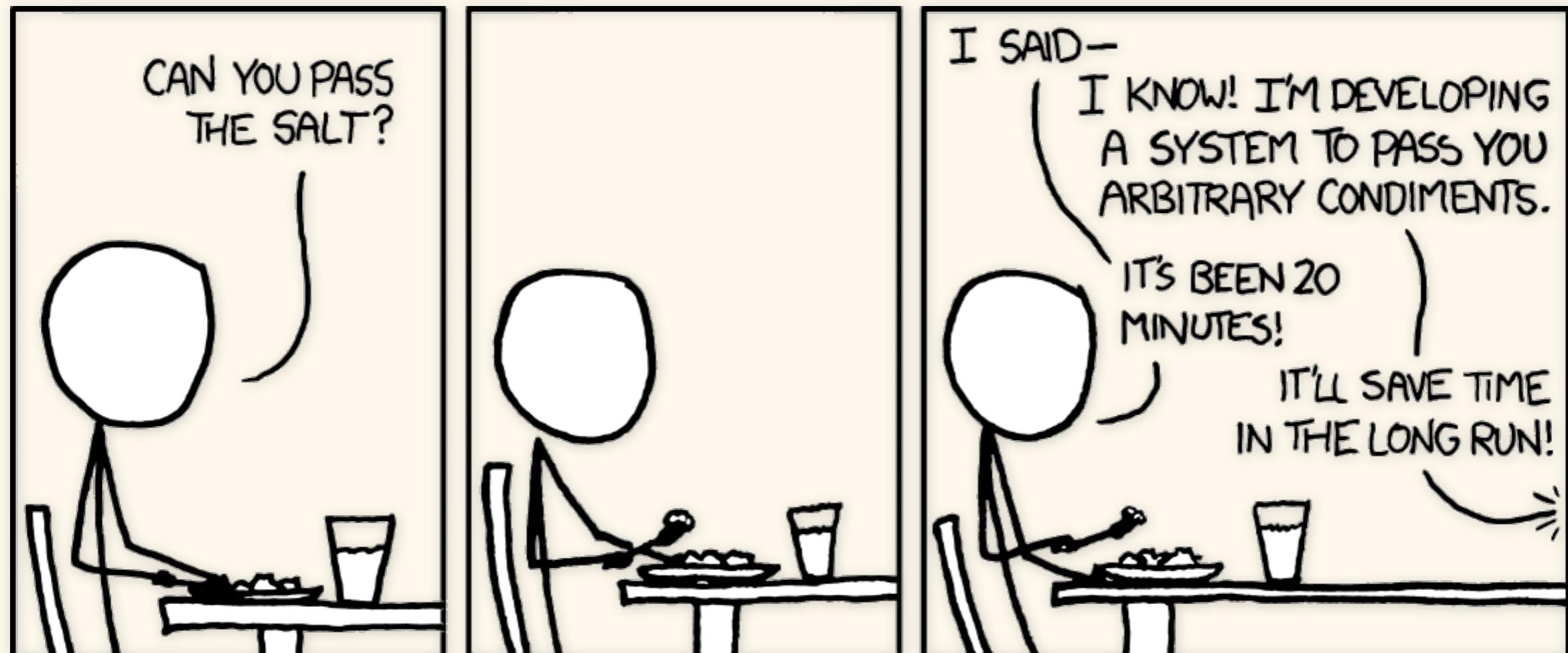
What actually happens



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

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

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

Triple quotes

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

Triple quotes

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

f-strings

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

f-strings

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

f-strings

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

f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num}")
The number is 33.33333333333336
```

f-strings

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

f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num:5.3}")
The number is 33.3
           1 2 3 4 5
```

f-strings

```
>>> num = 100 / 3
>>> print(f"The number is {num:05.3}")
The number is 033.3
           1 2 3 4 5
```

f-strings

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

f-strings

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

f-strings

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

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}>"""
```

Ternary operator

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

Ternary operator

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

Ternary operator

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

x if (condition) else y

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)
```

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)
```

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}>"""


```

Loop-else construct

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

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")
```

Built-in functions

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

Iterator wrappers

Don't:

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

Do:

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

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")
```

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'
```

Path manipulation

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

"The" line

```
from pathlib import Path
```

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()
```

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('.'))
```

Properties of Path object

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

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')
```

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 (...)
```

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()
```

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"): ...
```

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")
```

Moving and removing

```
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()
```

We all did this once

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

... but it's extremely dangerous

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

The rescue

import shutil

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")
```

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)
```

...

Temporary directories

Don't:

```
os.mkdir("temp")
```

```
...
```

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

Do:

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

```
...
```

collections – Container datatypes

- [defaultdict](#)
- [Counter](#)

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), ...]
```

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])
```

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'
```

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'
```

Need to check error?

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

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
```

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.
```

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")
```

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)
```

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)
```

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()
```

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'
```

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!

```
475 # Extract all sequence and chemical component details, and
476 # populate seq_to_structure_mappings using author chain IDs.
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(dict)
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     seq = []
485     all_chem_comp_info = []
486     for monomer_index, monomer in enumerate(seq_info):
487         all_chem_comp_info.append(chem_comp_info[monomer_index])
488         if "peptide" in chem_comp_info[monomer_index].type.lower():
489             code = PDBData.protein_letters_3to1.get(f'{monomer.id: <3}', "X")
490         elif (
491             "dna" in chem_comp_info[monomer_index].type.lower()
492             or "rna" in chem_comp_info[monomer_index].type.lower()
493         ):
494             code = PDBData.nucleic_letters_3to1.get(f'{monomer.id: <3}', "X")
495         else:
496             code = "X"
497             seq.append(code if len(code) == 1 else "X")
498     author_chain_to_sequence[author_chain] += "".join(seq)
499     all_chem_comp_details[author_chain].extend(all_chem_comp_info)
500     if current_mapping:
501         start_index = len(current_mapping)
502         current_mapping.update(
503             {
504                 start_index + value_index: value
505                 for value_index, value in enumerate(mmcif_current_mapping.values())
506             }
507         )
508     else:
509         current_mapping.update(mmcif_current_mapping)
```

Nope

```
475 # Extract all sequence and chemical component details, and
476 # populate seq_to_structure_mappings using author chain IDs.
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(dict)
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     seq = []
485     all_chem_comp_info = []
486     for monomer_index, monomer in enumerate(seq_info):
487         all_chem_comp_info.append(chem_comp_info[monomer_index])
488         if "peptide" in chem_comp_info[monomer_index].type.lower():
489             code = PDBData.protein_letters_3to1.get(f'{monomer.id: <3}', "X")
490         elif (
491             "dna" in chem_comp_info[monomer_index].type.lower()
492             or "rna" in chem_comp_info[monomer_index].type.lower()
493         ):
494             code = PDBData.nucleic_letters_3to1.get(f'{monomer.id: <3}', "X")
495         else:
496             code = "X"
497             seq.append(code if len(code) == 1 else "X")
498     author_chain_to_sequence[author_chain] += "".join(seq)
499     all_chem_comp_details[author_chain].extend(all_chem_comp_info)
500     if current_mapping:
501         start_index = len(current_mapping)
502         current_mapping.update(
503             {
504                 start_index + value_index: value
505                 for value_index, value in enumerate(mmcif_current_mapping.values())
506             }
507         )
508     else:
509         current_mapping.update(mmcif_current_mapping)
```

```
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(dict)
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     seq = []
485     all_chem_comp_info = []
486     for monomer_index, monomer in enumerate(seq_info):
487         all_chem_comp_info.append(chem_comp_info[monomer_index])
488         if "peptide" in chem_comp_info[monomer_index].type.lower():
489             code = PDBData.protein_letters_3to1.get(f'{monomer.id: <3}', "X")
490         elif (
491             "dna" in chem_comp_info[monomer_index].type.lower()
492             or "rna" in chem_comp_info[monomer_index].type.lower()
493         ):
494             code = PDBData.nucleic_letters_3to1.get(f'{monomer.id: <3}', "X")
495         else:
496             code = "X"
497             seq.append(code if len(code) == 1 else "X")
498     author_chain_to_sequence[author_chain] += "".join(seq)
499     all_chem_comp_details[author_chain].extend(all_chem_comp_info)
500     if current_mapping:
501         start_index = len(current_mapping)
502         current_mapping.update(
503             {
504                 start_index + value_index: value
505                 for value_index, value in enumerate(mmcif_current_mapping.values())
506             }
507         )
508     else:
509         current_mapping.update(mmcif_current_mapping)
```

```
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(dict)
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     seq = []
485     all_chem_comp_info = []
486     for monomer_index, monomer in enumerate(seq_info):
487         all_chem_comp_info.append(chem_comp_info[monomer_index])
488         if "peptide" in chem_comp_info[monomer_index].type.lower():
489             code = PDBData.protein_letters_3to1.get(f'{monomer.id: <3}', "X")
490         elif (
491             "dna" in chem_comp_info[monomer_index].type.lower()
492             or "rna" in chem_comp_info[monomer_index].type.lower()
493         ):
494             code = PDBData.nucleic_letters_3to1.get(f'{monomer.id: <3}', "X")
495         else:
496             code = "X"
497         seq.append(code if len(code) == 1 else "X")
498     author_chain_to_sequence[author_chain] += "".join(seq)
499     all_chem_comp_details[author_chain].extend(all_chem_comp_info)
500     if current_mapping:
501         start_index = len(current_mapping)
502         current_mapping.update(
503             {
504                 start_index + value_index: value
505                 for value_index, value in enumerate(mmcif_current_mapping.values())
506             }
507         )
508     else:
509         current_mapping.update(mmcif_current_mapping)
```

To single letter

```
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(dict)
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     if current_mapping:
489         start_index = len(current_mapping)
490         current_mapping.update(
491             {
492                 start_index + value_index: value
493                 for value_index, value in enumerate(mmcif_current_mapping.values())
494             }
495         )
496     else:
497         current_mapping.update(mmcif_current_mapping)
```

```
477 author_chain_to_sequence = defaultdict(str)
478 all_chem_comp_details = defaultdict(list)
479 seq_to_structure_mappings = defaultdict(dict)
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     if current_mapping:
489         start_index = len(current_mapping)
490         current_mapping.update(
491             {
492                 start_index + value_index: value
493                 for value_index, value in enumerate(mmcif_current_mapping.values())
494             }
495         )
496     else:
497         current_mapping.update(mmcif_current_mapping)
```

Why not list?

```
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())
```

```
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())
```

```
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
483     chain_to_sequence[author_chain] += seq_info
484     chain_to_chem_comp[author_chain].extend(chem_comp_info)
485
486     mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
487     chain_to_structure[author_chain].extend(mmcif_current_mapping.values())
488
```

```
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
483     chain_to_sequence[author_chain] += seq_info
484     chain_to_chem_comp[author_chain].extend(chem_comp_info)
485
486     mmcif_current_mapping = mmcif_seq_to_structure_mappings[chain_id]
487     chain_to_structure[author_chain].extend(mmcif_current_mapping.values())
488
```

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 pdble`
- 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

Code, code, code

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

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](#)

Q&A

Thank you for Listening

Appendix: SOLID principles of OOP

- The Single-responsibility principle
 - The Open-closed principle
 - The Liskov substitution principle
 - The Interface segregation principle
 - The Dependency 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, ...)
```

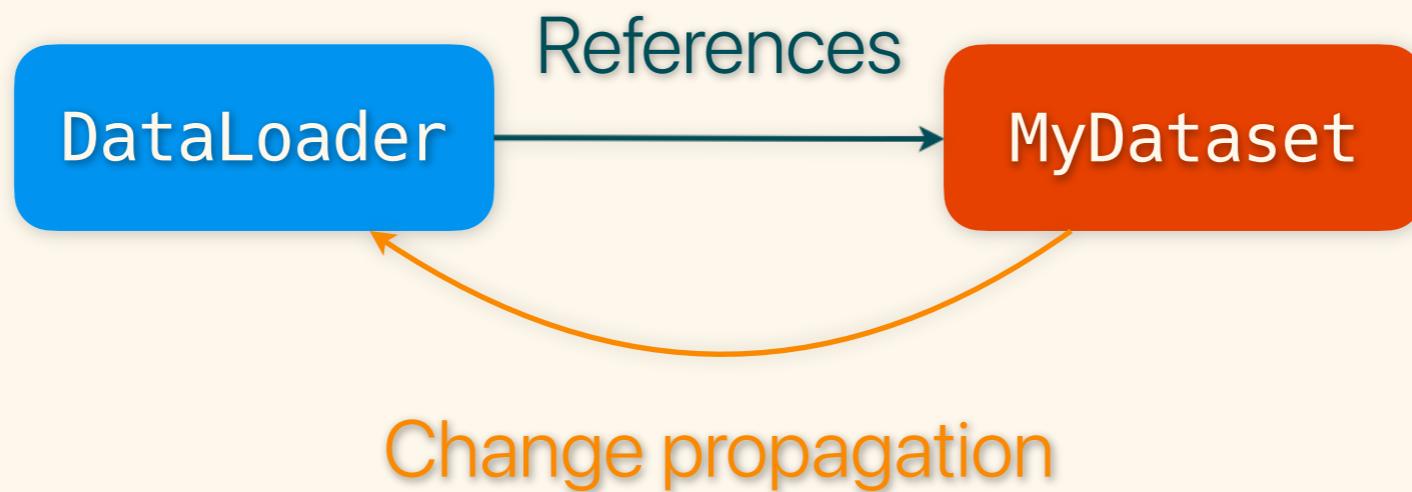
Single-responsibility principle

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

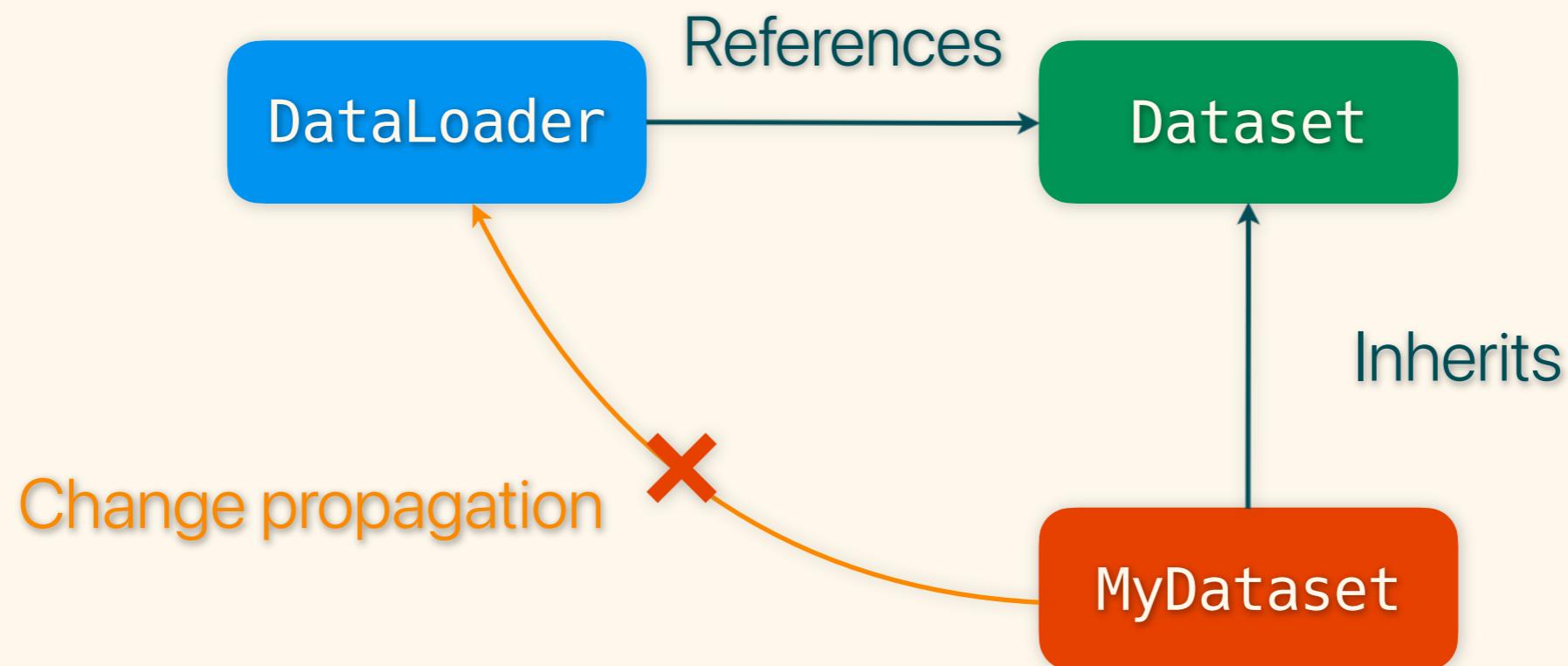
...

```
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

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

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

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

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): ...
```
