

§ 14 PoC 1 結果報告 — Any 含む Python の Isolate bit-exact

項目	値
対象 spec	spec_section_14_draft.md v0.1, § 14.9 PoC 計画 PoC 1
実施日	2026-06-04
環境	WSL Ubuntu 24.04 / Rust 1.93.1 / CPython 3.12.3 / PyO3 0.22.6
目標	§ 13 で DEFER_HARD に分類されている Any を含む実 Python サンプルを 5 件、Deterministic Python Isolate 内で実行し、純 Python 実行と stdout SHA-256 が完全一致するか実機検証
結果	5/5 PASS(bit-exact 完全一致)

0. 結論

Any を含む 5 サンプル全件で Python(リファレンス)= Rust + PyO3 + CPython-Isolate の出力 SHA-256 が完全一致。§ 14 の主張「動的型を Rust で再発明せず、Python のまま deterministic CPython に隔離実行することで bit-exact を保つ」が、Any 領域(§ 13 で reject されていた強い動的構文)で実機成立することを確認しました。

1. 検証サンプル(各 § 13 で DEFER_HARD)

#	path	構文	概要
01	samples/any01_param_type.py	def f(x: Any) -> int: + isinstance dispatch	int / str / list / dict / bool / float への分岐処理
02	samples/any02_heterogeneous_list.py	list[Any] + type(x).__name__	7 種混在 list の型名反射
03	samples/any03_dict_str_any_dispatch.py	dict[str, Any] + sorted key 反復	5 種値型の per-key 分岐フォーマット
04	samples/any04_any_return.py	def f(idx: int) -> Any: + str() coerce	異種返値の str 化
05	samples/any05_polymorphic_add.py	def add(a: Any, b: Any) -> Any:	int/str/list/float/tuple での + 多態

2. 実行結果

```
=== build runner ===
  Compiling poc1_any v0.1.0
  Finished `release` profile [optimized] target(s) in 7.46s
binary: 338568 bytes

=== run + bit-exact compare (5 Any samples) ===
any01_param_type          PASS  sha=26d418acf424cc56... (30B)
any02_heterogeneous_list  PASS  sha=7447433c14a91c01... (66B)
any03_dict_str_any_dispatch PASS  sha=9d2c3df815859c0b... (61B)
any04_any_return          PASS  sha=ef68b7955ac719df... (41B)
any05_polymorphic_add     PASS  sha=a492171f7eb1750a... (85B)

=== summary: PASS=5 / 5 ===
```

各 sample の出力サンプル:

#	出力(共通、両系統で完全一致)
01	results=[84, 5, 15, 2, 1, -1]
02	types=['int', 'str', 'float', 'list', 'dict', 'bool', 'NoneType']
03	active=b:True\ count=i:42\ items=l:3\ name=s:alpha\ ratio=f:0.75
04	42\ hello\ [1, 2, 3]\ {'x': 1, 'y': 2}\ True
05	int_add=30\ str_add=foobar\ list_add=[1, 2, 3, 4]\ float_add=4.0\ tuple_add=(1, 2, 3, 4)

3. アーキテクチャ — 汎用 Isolate runner

PoC 0 と異なり、PoC 1 では 汎用 Isolate runner(任意の .py を引数で受けて実行)を構成:

```
fn main() {
  env::set_var("PYTHONHASHSEED", "0");
  env::set_var("PYTHONUNBUFFERED", "1");

  let path = env::args().nth(1).expect("usage: poc1_isolate <script.py>");
  let code = fs::read_to_string(&path).expect("read failed");

  Python::with_gil(|py| {
    py.run_bound(&code, None, None).expect("Python error");
    // Explicit flush - pip buffer to OS stdout fd before Isolate teardown
    let _ = py.run_bound("import sys\nsys.stdout.flush()\nsys.stderr.flush()",
      None, None);
  });
}
```

```
});  
}
```

- `PYTHONHASHSEED=0` で hash 順序固定
- `PYTHONUNBUFFERED=1` + 明示 `sys.stdout.flush()` で出力 buffering 取りこぼし回避(初回実行で 0 B 出力になる現象を解決)
- Isolate からは Python ソース文字列をそのまま `run_bound` に渡す。AST / Slot IR を通らず、CPython が直接実行

4. binary 実測

軸	値
Rust binary	338,568 bytes(~331 KB)
依存リンク	<code>libpython3.12.so.1.0</code> (動的)、 <code>libc.so.6</code> 、 <code>libm.so.6</code>
Cargo deps	<code>pyo3 = { version = "0.22", features = ["auto-initialize"] }</code> のみ(他ゼロ)
build 時間	初回 ~30 秒、増分 7-8 秒

PoC 0(348 KB)と同水準。サンプルごとに別バイナリを作らず、ひとつの runner で 5 samples すべてカバー。

5. 検証された § 14 要素

§ 14 要素	検証状況
Any の Dynamic Isolate 委譲	☑5/5 PASS
多態 <code>__add__</code> の Isolate 実行	☑#05 で int/str/list/float/tuple すべて
isinstance dispatch	☑#01 / #03
<code>type().__name__</code> 反射	☑#02
<code>dict[str, Any]</code> の sorted iteration	☑#03
Any 戻り値の <code>str()</code> coerce	☑#04
<code>PYTHONHASHSEED</code> 決定論モード	☑ <code>env::set_var</code> 前置で適用
stdout flush 経路の確定	☑ <code>PYTHONUNBUFFERED</code> + 明示 flush
Python ソースを断片のまま渡す	☑ <code>py.run_bound(&code, ...)</code> で直接実行

6. 重要な観察

1. PyO3 経由 CPython の決定論は素直: 同一スクリプトを `python3` 直接実行と Rust+PyO3 で走らせると、同じ `stdout` バイト列になる。CPython 3.12.3 という単一実装で動くため、両者で異なる挙動を取りようがない(libm 差分等を含まない単純動的 Python の場合)。
2. `buffering trap` が一つあった: 初回実行で Rust binary 側 `stdout` が 0 B(buffer flush されず process exit)。PYTHONUNBUFFERED=1 を env に追加 + 明示 `sys.stdout.flush()` で解消。これは § 14 spec の `Isolate runtime` 設計に「`stdout flush` を `Isolate teardown` 前に必ず実行」を明文化すべき学び。
3. `Any` は CPython にとって型注釈に過ぎない: 実行時は単に動的 dispatch。Hybrid 側からは「型情報を捨てて CPython に丸投げ」しているだけで、CPython 自身は型に関知せずに動く。これが「意味の正本は CPython に置く」哲学の実体。
4. 5 samples で 1 件も `buffering` 以外のズレが出なかった: 「Hybrid Isolate は安全」のエビデンスとして強い。少なくとも `Any` 領域では PoC レベルで bit-exact が成立する。

7. 配布物

```
engine/poc1_any/
├─ Cargo.toml                pyo3 0.22 + auto-initialize
├─ src/main.rs               汎用 Isolate runner
├─ samples/
│   ├─ any01_param_type.py   (DEFER_HARD: Any 関数引数)
│   ├─ any02_heterogeneous_list.py (DEFER_HARD: list[Any])
│   ├─ any03_dict_str_any_dispatch.py (DEFER_HARD: dict[str, Any])
│   ├─ any04_any_return.py   (DEFER_HARD: Any 戻り値)
│   └─ any05_polymorphic_add.py (DEFER_HARD: 多態 +)
├─ run_bit_exact.sh          ビルド→5件実行→SHA-256比較
└─ target/release/poc1_isolate 338,568 B Isolate runner
```

8. § 14.9 PoC ロードマップ 進捗

Phase	内容	状態
☑PoC 0	最小実証(static + 1 isolate call で bit-exact)	完了 (POC0_REPORT_2026-06-04.md)
☑PoC 1	<code>Any</code> を含む 5 sample を Isolate 化	完了(本書)
PoC 2	<code>*args/**kwargs</code> + 動的 <code>getattr</code> + 動的 monkey patching 各 5 sample	次
PoC 3	partition(関数単位)の自動振り分け実装、混在サンプル 10 件	

Phase	内容	状態
PoC 4	musl-libm pin、cross-platform 検証	
PoC 5	RustPython 版 Light tier 同型サ ンプル、差分計測	

PoC 1 完了。§ 13 で DEFER_HARD だった **Any** カテゴリが Hybrid Isolate で受理・bit-exact 動作することを 5/5 で実機実証。PoC 2 へ進行可。