Queens College of CUNY
Department of Computer Science
Programming Languages
(CSCI 316)
Winter 2026

Assignment #5
"Variables and their Attributes"
Due: January 10, 2026

Introduction:

In this assignment, we use Python code to summarize information about the variables and their primary attributes: name, type, address, value, location, scope, and lifetime, as well as reference count. We "dump" these variables at different points to illustrate how variables go in and out of scope and how the attributes change over the course of the execution.,

Submissions:

In the Google form, please submit:

- Assignment05.py (source code)
- Assignment05.txt (console output)

Tasks:

[0] Create a new program Assignment04.py, update the assignment name and number in the comment at the top, and perform any other preliminary tasks associated with a new assignment.

[1] Use the following code and get it to run for the provided demo data and review the table of variables.

[2] Add two recursive functions to the code: one that computes a recursive formula and another that performs an algorithm like searching or sorting. For searching or sorting, you will need either a sorted or unsorted array/list as input.

[3] Call dump_variables() before, during, and immediately after each of the two functions you chose. When you call those two recursive functions, keep the input small as otherwise the output will get unwieldly.

[4] Answer this question in free-from comments in your output: which variable attributes are changing and which are not as the code progresses.

```
import gc
import inspect
import sys
from collections import deque
from types import FrameType

def infer_scope(name: str, frame: FrameType) -> str:
    if name in frame.f_locals:
        return "local"
    if name in frame.f_globals:
        return "global"
    if name in frame.f_builtins:
        return "builtin"
    return "unknown"
```

```
def infer_lifetime(scope: str) -> str:
    if scope == "local":
        return "until function returns"
    if scope == "global":
        return "until program exit"
    if scope == "builtin":
        return "interpreter lifetime"
    return "unknown"

def safe_repr(x, limit=80) -> str:
    try:
        r = repr(x)
    except Exception as e:
        return f"<repr error: {e.__class__.__name__}>"
    return r if len(r) <= limit else r[: limit - 3] + "..."

def safe_refcount(x) -> str:
    try:
        return str(sys.getrefcount(x) - 1)  # subtract temp ref
    except Exception as e:
        return f"err:{e.__class__.__name__}"

def safe_is_tracked(x) -> str:
    try:
        return "yes" if gc.is_tracked(x) else "no"
    except Exception as e:
        return f"err:{e.__class__.__name__}"

def is_reachable_from_roots(target,
                            frame: FrameType,
                            max_nodes: int = 20000,
                            max_depth: int = 12) -> str:
    """
    Approx reachability via bounded BFS over gc referents from roots:
    locals/globals/builtins + sys.modules.
    """
    try:
        tgt_id = id(target)
    except Exception:
        return "UNKNOWN(unid)"

    roots = [frame.f_locals, frame.f_globals, frame.f_builtins, sys.modules]

    q = deque()
    seen = set()

    def enqueue(obj, depth):
        oid = id(obj)
        if oid in seen:
            return
        seen.add(oid)
        q.append((obj, depth))

    for r in roots:
        enqueue(r, 0)
```

```
    if tgt_id in seen:
        return "YES"

    nodes = 0
    while q:
        obj, depth = q.popleft()
        nodes += 1
        if nodes > max_nodes:
            return "UNKNOWN(limit)"
        if depth >= max_depth:
            continue

        try:
            referents = gc.get_referents(obj)
        except Exception:
            continue

        for ref in referents:
            rid = id(ref)
            if rid == tgt_id:
                return "YES"
            if rid not in seen:
                enqueue(ref, depth + 1)

    return "NO"

def dump_variables(frame=None, *,
                   include_builtins=False,
                   reachability=True,
                   max_nodes=20000,
                   max_depth=12,
                   value_limit=80):
    """
    Dumps visible variables (globals+locals) with attributes including VALUE.
    """
    if frame is None:
        frame = inspect.currentframe().f_back

    merged = {}
    merged.update(frame.f_globals)
    merged.update(frame.f_locals)

    if include_builtins:
        merged.update(frame.f_builtins)

    # Header
    cols = [
        ("Name", 20),
        ("Type", 16),
        ("Address", 18),
        ("Scope", 9),
        ("Lifetime", 22),
        ("RefCt", 6),
        ("GC", 4),
```

```
    ]
    if reachability:
        cols.append(("Reach", 14))
    cols.append(("Value", 0))

    line = "=" * 140
    print(line)
    header = ""
    for c, w in cols:
        header += (f"{c:<{w}} " if w > 0 else c)
    print(header.rstrip())
    print(line)

    for name, value in sorted(merged.items(), key=lambda kv: kv[0]):
        tname = type(value).__name__
        addr = hex(id(value))
        scope = infer_scope(name, frame)
        life = infer_lifetime(scope)
        refc = safe_refcount(value)
        tracked = safe_is_tracked(value)
        val = safe_repr(value, limit=value_limit)

        if reachability:
            reach = is_reachable_from_roots(
                value, frame, max_nodes=max_nodes, max_depth=max_depth
            )
            row = (f"{name:<20} {tname:<16} {addr:<18} {scope:<9} {life:<22} "
                   f"{refc:<6} {tracked:<4} {reach:<14} {val}")
        else:
            row = (f"{name:<20} {tname:<16} {addr:<18} {scope:<9} {life:<22} "
                   f"{refc:<6} {tracked:<4} {val}")

        print(row)

    print(line)

GLOBAL_VAR = {"k": "v"}

def main():
    local_list = [1, 2, 3]
    local_dict = {"a": 1, "b": 2}
    temp = "hello"

    dump_variables(value_limit=70)

if __name__ == "__main__":
    main()
```