Weekly Homework — Week 04, Term 1

PM4 Test Prep — File Formats & Data Linking Practice Problems

Programming Module 4 | Applied Computing Australia

Name		Date	

Part A: Predict the Output

What will each code snippet print? Write your answer before checking. Trace through the code line by line.

A1. strip() and split("|")

```
line = "VIC|0.282|1.20\n"
parts = line.strip().split("|")
print(parts)
print(len(parts))
```

Output:

A2. Parsing a pipe-delimited file

```
lines = [
    "state|rate|tariff\n",
    "VIC|0.282|0.05\n",
    "NSW|0.305|0.05\n",
]
for line in lines[1:]:
    parts = line.strip().split("|")
    print(f"{parts[0]}: ${parts[1]}/kWh")
```

Output:

A3. Building an XML string

```
brand = "SunPower"
watts = 440
xml = f"  <panel>\n"
xml += f"    <brand>{brand}</brand>\n"
xml += f"    <watts>{watts}</watts>\n"
xml += f"  </panel>\n"
print(xml)
```

Output:

A4. Data linking — match found

```
panels = [
    {"panel_id": 1, "brand": "SunPower"},
    {"panel_id": 2, "brand": "Jinko"},
    {"panel_id": 3, "brand": "LONGi"},
]

def find_panel(panels, target_id):
    for panel in panels:
        if panel["panel_id"] == target_id:
            return panel
    return None

result = find_panel(panels, 2)
print(result["brand"])
```

Output:

A5. Data linking — no match

```
panels = [
    {"panel_id": 1, "brand": "SunPower"},
    {"panel_id": 2, "brand": "Jinko"},
]

def find_panel(panels, target_id):
    for panel in panels:
        if panel["panel_id"] == target_id:
            return panel
    return None

result = find_panel(panels, 5)
if result is None:
    print("Panel not found")
else:
    print(result["brand"])
```

Output:

A6. Tricky — strip() vs no strip()

```
line = "SA|0.345|0.04\n"

# Without strip()
parts_raw = line.split("|")
print(repr(parts_raw[-1]))

# With strip()
parts_clean = line.strip().split("|")
print(repr(parts_clean[-1]))
```

Output:

Part B: Find the Error

Each snippet has a bug. Identify the error and explain the fix.

B1. Forgetting strip()

```
line = "QLD|0.268|0.06\n"
parts = line.split("|")
tariff = float(parts[2])
print(f"Feed-in tariff: {tariff}")
```

Error:

Fix:

B2. Missing closing XML tag

```
xml = "<panels>\n"
xml += "  <brand>SunPower</brand>\n"
xml += "  <watts>440</watts>\n"
print(xml)
```

Error:

Fix:

B3. Wrong split delimiter

```
line = "VIC|0.282|1.20\n"
parts = line.strip().split(",")
print(parts[0])
print(parts[1])
```

Error:

Fix:

B4. Not handling None from lookup

```
panels = [{"panel_id": 1, "brand": "SunPower"}]

def find_panel(panels, target_id):
    for panel in panels:
        if panel["panel_id"] == target_id:
            return panel
    return None

result = find_panel(panels, 99)
print(result["brand"])
```

Error:

Fix:

B5. Forgetting to skip the header

```
lines = [
    "state|rate|tariff\n",
    "VIC|0.282|0.05\n",
    "NSW|0.305|0.05\n",
]
for line in lines:
    parts = line.strip().split("|")
    rate = float(parts[1])
    print(f"Rate: {rate}")
```

Error:

Fix:

Part C: Fill in the Blanks

Complete the code by filling in each blank (_____).

C1. Reading and parsing a pipe-delimited file

```
rates = []
with open("electricity_rates.txt", "r") as f:
    lines = f._____()

    for line in lines[_____:]:
        parts = line._____().split("_____")
        record = {
            "state": parts[0],
            "rate": float(parts[1])
        }
        rates.append(record)

print(f"Loaded {len(rates)} rates")
```

C2. Building XML output with f-strings

```
panels = [
    {"brand": "SunPower", "watts": 440},
    {"brand": "Jinko", "watts": 420},
]

xml = '<?xml version="1.0" encoding="UTF-8"?>\n'
xml += '_____\n'

for panel in panels:
    xml += f'  <panel>\n'
    xml += f'    <_____>{panel["brand"]}</_____>\n'
    xml += f'    <watts>{panel["_____"]}</watts>\n'
    xml += f'  </panel>\n'

xml += '_____\n'
print(xml)
```

C3. Data linking lookup function

```
def find_installer(installers, target_id):
    for installer in installers:
        if installer["_____"] == _____:
            return _____
    return _____

# Usage
result = find_installer(installers, 3)
if result is _____:
    print("Not found")
else:
    print(result["name"])
```

Part D: Write the Code

Write complete programs from scratch. Use correct indentation and snake_case variable names.

D1. Parse a pipe-delimited file into a list of dicts

The file `solar_installers.txt` contains pipe-delimited data with a header line:

```
installer_id|name|state|rating
1|SolarMax|VIC|4.8
2|GreenEnergy|NSW|4.5
3|SunFirst|QLD|4.9
```

Write code to read this file, skip the header, parse each line into a dictionary with keys `"installer_id"` (int), `"name"` (str), `"state"` (str), `"rating"` (float), and append each dict to a list called `installers`. Then print how many installers were loaded.

D2. Write a list of dicts to XML format

Given this list of solar panel data:

```
panels = [
    {"brand": "SunPower", "watts": 440, "efficiency": 22.8},
    {"brand": "Jinko", "watts": 420, "efficiency": 21.5},
    {"brand": "LONGi", "watts": 425, "efficiency": 22.0},
]
```

Write code that builds an XML string with declaration `<?xml version="1.0" encoding="UTF-8"?>`, root element `<solar_panels>`, and one `<panel>` child per item containing `<brand>`, `<watts>`, and `<efficiency>` tags. Write the result to `"panels_export.xml"`.

D3. Link two datasets by shared ID

Given these two lists:

```
panels = [
    {"panel_id": 1, "brand": "SunPower", "watts": 440},
    {"panel_id": 2, "brand": "Jinko", "watts": 420},
    {"panel_id": 3, "brand": "LONGi", "watts": 425},
]
installations = [
    {"install_id": 1, "panel_id": 2, "customer": "Smith", "quantity": 20},
    {"install_id": 2, "panel_id": 1, "customer": "Jones", "quantity": 15},
    {"install_id": 3, "panel_id": 3, "customer": "Lee", "quantity": 22},
]
```

Write a `find_panel(panels, target_id)` function that returns the matching panel dict (or `None`). Then loop through installations, look up each panel, and print the customer name, panel brand, and total wattage (watts × quantity).

Part E: File Format Comparison Table

Fill in the missing cells. Choose from the options provided below the table.

	Pipe-delimited TXT	**CSV**	**XML**
Delimiter / Structure			
Python read method			
Header handling			
Best for			
Can nest data?			
Human-readable?			

Options: `|` pipe character; `,` comma; `<tag>...</tag>` matching tags; `split("|")` with `strip()`; `csv.reader()`; f-strings building tag strings; skip `lines[0]` or `lines[1:]`; first row is column names; declaration + root element; flat tabular data; standard tabular data (spreadsheets); hierarchical/nested data; No; No; Yes; Yes (compact); Yes (compact); Yes (verbose)