Container data and Actions

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.
Design of a solution
Structured Data

If you have lots and lots of one type of data (for example, the ages of all the people in Turkey):

- You can store them into memory consecutively (supported by most PLs)
  - This is called *arrays*.
- Easy to access an element. Nth element:
  - \(<\text{Starting-address}>+ (N-1)\>*<\text{Word Width}>\)
  - Ex: 2nd element is at 128 + (2-1) * 4 = 132
Structured Data

- What if you have to make a lot of deletions and insertions in the middle of an array?
- Then, you have to store your data in blocks/units such that each unit has the starting address of the next unit/block.
Containers

- Containers of basic data:
  - Strings
  - Tuples
  - Lists
Strings

- Sequence of characters:
  - Ex: “Book”, “Programming”, “Python”

- How can they be represented?
  1. Put a set of characters one after the other and end them with a non-character value.
  2. At the beginning of the characters, specify how many characters follow.

- Both have advantages and disadvantages.
Strings in Python

Python provides the `str` data type for strings:

```python
>>> "Hello?"
'Hello?'
>>> type("Hello?")
<type 'str'>
```

- Simplest operation with a string:

```python
>>> len("Hello?")
6
```
Strings in Python

- Accessing elements of a string
  - “Hello?”[0] → 1\textsuperscript{st} character (i.e., “H”)
  - “Hello?”[4] → 5\textsuperscript{th} character

- Indexing starts at 0!!!

- What is the last element then?
  - “Hello?”[len(“Hello?”) - 1]

- Negative indexing possible:
  - Last element: “Hello?”[-1] → “?”

- In general:
  - String[start:end:step]
  - Ex: “Hello?”[0:4:2] → “Hi”
Creating Strings in Python

1. Enclosing a set of characters between quotes:
   - “ali”, “veli”, “deli”, ...

2. Using the `str()` function:
   - `str(4.5) → “4.5”`

3. Using the `raw_input()` function:

```python
>>> a = raw_input("--> ")
--> Do as I say
>>> a
’Do as I say’
>>> type(a)
<type ’str’>
```
Tuples

Tuple: ordered set of data:

- (1, 2, 3)
- ("a", "b", "c")

May be heterogeneous:

Tuples in Python

Tuples in Python: collection of data enclosed in parentheses, separated by comma.

- Accessing elements of a tuple (like strings):
  - Positive Indexing: `(1, 2, 3, 4, "a")[2]` returns 3.
  - Negative Indexing: `(1, 2, 3, 4, "a")[-1]` returns 'a'.
  - Ranged Indexing, *i.e.*, `[start:end:step]`: `(1, 2, 3, 4, "a")[0:4:2]` leads to `(1, 3)`.

```python
>>> (1, 2, 3, 4, "a")
(1, 2, 3, 4, 'a')
>>> type((1, 2, 3, 4, "a"))
<type 'tuple'>
```
Creating Tuples in Python

1. Enclosing data within parentheses:
   - Ex: `(1, "a", "cde", 23)`

2. Using the `tuple()` function:
   - Ex: `tuple("ABC") → ('A', 'B', 'C')`

3. Using the `input()` function:

```python
>>> a = input("Give me a tuple:")
Give me a tuple:(1, 2, 3)
>>> a
(1, 2, 3)
>>> type(a)
<type 'tuple'>
```
Lists

- Similar to tuples.

- Difference:
  - Tuples are immutable (i.e., not changeable) whereas lists are mutable.
Lists in Python

- Lists in Python: collection of data enclosed in brackets, separated by comma.
- Accessing elements of a list (like strings & tuples):
  - Negative Indexing: `[1, 2, 3, 4, "a"] [-1] returns 'a'.
  - Ranged Indexing, i.e., `[start:end:step]`: `[1, 2, 3, 4, "a"] [0:4:2] leads to [1, 3].
Creating Lists in Python

1. Enclosing data within brackets:
   - Ex:  [1, “a”, “cde”, 23]

2. Using the list() function:
   - Ex:  list(“ABC”) → [‘A’, ‘B’, ‘C’]

3. Using the range() function:
   - Ex: range(1, 10, 2) → [1, 3, 5, 7, 9]

4. Using the input() function:

   >>> a = input("Give me a list:")
   Give me a list:[1, 2, "a"]
   >>> a
   [1, 2, ’a’]
   >>> type(a)
   <type ’list’>
Modifying a List in Python

- List[range] = Data
  
  Ex:

  >>> L = [3, 4, 5, 6, 7, '8', 9, '10']
  >>> L[:2]
  [3, 5, 7, 9]
  >>> L[:2] = [4, 6, 8, 10]
  >>> L[:2]
  [4, 6, 8, 10]
  >>> L[]
  [4, 4, 6, 6, 8, '8', 10, '10']

- Using the append() function:
  - List.append(item)
  - Ex: [1, 2, 3].append(5) → [1, 2, 3, 5]
Modifying a List in Python

- Using the extend() function:
  - `List.extend(Another_list)`
  - Ex: `[1, 2, 3].extend(["a", "b"]) \rightarrow [1, 2, 3, "a", "b"]`

```python
>>> L.extend(["a", "b"])
>>> L
[4, 4, 6, 6, 8, '8', 10, '10', 'a', 'a', 'b']
```

- Using the insert() function:
  - `List.insert(index, item)`
```python
>>> L=[1, 2, 3]
>>> L
[1, 2, 3]
>>> L.insert(1, 0)
>>> L
[1, 0, 2, 3]
```
Removing Elements from a List in Python

- **del statement**: `del L[start:end]`

  ```python
define L
[1, 0, 2, 3]
define del L[1]
define L
[1, 2, 3]
```

- **L.pop() function**: `L.pop([index])`

  ```python
define L=[1,2,3]
define L.pop()
define L
[1, 2]
define L.pop(0)
define L
[1]
define L.pop()
define L
[2]
```

- **L.remove() function**: `L.remove(value)`

  ```python
define L
[2, 1, 3]
define L.remove(1)
define L
[2, 3]
```
Accessing Data/Containers by Names: Variables

■ Naming:
  ▪ Usually: A combination of letters and numbers
  ▪ Ex: a123, 123a, ...

■ Scope & Extent:
  ▪ Scope: Where a variable can be accessed.
  ▪ Extent: The lifetime of a variable.

■ Typing:
  ▪ Statically typed: The type of a variable is fixed.
  ▪ Dynamically typed: The type of a variable is variable 😊
Today

- Container/structured data
  - Variables
  - Aliasing problem
  - Actions

- Reminder:
  - Tentative midterm date: 9 December at 17:40.
Variables in Python

- We don’t need to define a variable before using it.
- We don’t need to specify the type of a variable.

```
>>> a = 4
>>> b = 3
>>> c = a + b
>>> a
4
>>> b
3
>>> c
7
```

- ‘=’ means “Change the content of the variable with the value at the right-hand side”.
  - Assignment!
- The left-side of the assignment should be a valid variable name:
  - Ex: a+2 = 5 → NOT VALID!
Variable Naming in Python

- Variable names are case sensitive. So, the names `a` and `A` are two different variables.

- Variable names can contain letters from the English alphabet, numbers and an underscore `_`.

- Variable names can only start with a letter or an underscore. So, `10a`, `$a`, and `var$` are all invalid whereas `_a` and `a_20`, for example, are valid names in Python.

Variable names cannot be one of the keywords in Python:

- `and`, `del`, `from`, `not`, `while`,
- `as`, `elif`, `global`, `or`, `with`,
- `assert`, `else`, `if`, `pass`, `yield`,
- `break`, `except`, `import`, `print`,
- `class`, `exec`, `in`, `raise`,
- `continue`, `finally`, `is`, `return`,
- `def`, `for`, `lambda`, `try`
More on Variables in Python

• Typing of variables:
  – Python is dynamically typed:

```python
>>> a = 3
>>> type(a)
<type 'int'>
>>> a = 3.4
>>> type(a)
<type 'float'>
```

- Using variables:
Variables, Values and Aliasing in Python

Every data (whether constant or not) has an identifier (an integer) in Python:

```
>>> a = 1
>>> b = 1
>>> id(1)
135720760
>>> id(a)
135720760
>>> id(b)
135720760
```

• If the type of the data is mutable, there is a problem!!!

```
>>> a = [‘a’, ‘b’]
>>> b = a
>>> id(a)
3083374316L
>>> id(b)
3083374316L
>>> b[0] = 0
>>> a
[0, ‘b’]
```

This is called Aliasing.
Now

- We start another ingredient of a program:
  - Actions!
What are actions?

Actions in a PL are the *things* that we can do with the data. What could they be?

- Create data or modify data
- Interact with the external environment
Actions for creating/modifying data

- Evaluating a mathematical expression
  - But there are differences to the expressions in Mathematics
- Working with structured data
- Storing results of computations (in another data)
- Making a decision about how to proceed with the computation
  
  - if x*y < 3.1415 then \textit{do some action}
  - if "ali" in class_111_list then \textit{do some action}
  - if tall("ali") then \textit{do some action}
Interaction-type actions

- “Interaction” means Input/Output.
- Why interact with the environment? Why do we have Input/Output actions?
  - To react on a change in the external environment
  - To produce an effect in the external environment
Action Types in High-Level Languages

- Expression evaluation
  - $3 + 4 \times 5 / 2$

- Statement execution
  - `del L[2:4]`
An expression is a calculation which has a set of operations.

Operations have operators and operands.

Example: 3 + 4

- \(+\) → operator
- 3, 4 → operands

Figure 3.1: N-ary operation