15-110 Principles of Computing – F19

Lecture 11:
Tuples, Lists 2

Teacher:
Gianni A. Di Caro

So far about Python ...

- Basic elements of a program:
  - Literal objects
  - Variables objects
  - Function objects
  - Commands
  - Expressions
  - Operators

- Utility functions (built-in):
  - `print(arg1, arg2, …)`
  - `type(obj)`
  - `id(obj)`
  - `int(obj)`
  - `float(obj)`
  - `bool(obj)`
  - `str(obj)`
  - `input(msg)`
  - `len(non_scalar_obj)`

- Object properties
  - Literal vs. Variable
  - Type
  - Scalar vs. Non-scalar
  - Immutable vs. Mutable

- Conditional flow control

  -
    ```
    if cond_true:
        do_something
    ```

  -
    ```
    if cond_true:
        do_something
    else:
        do_something_else
    ```

  -
    ```
    if cond1_true:
        do_something_1
    elif cond2_true:
        do_something_2
    else:
        do_something_else
    ```

- Data types:
  - `int`
  - `float`
  - `bool`
  - `str`
  - `None`
  - `tuple`
  - `list`

- Relational operators
  - `>`
  - `<`
  - `>=`
  - `<=`
  - `==`
  - `!=`

- Logical operators
  - `and`
  - `or`
  - `not`

- Operators:
  - `=`
  - `+`
  - `+=`
  - `-`
  - `/`
  - `*`
  - `*=`
  - `//`
  - `%`
  - `**`
  - `[]`
  - `[:]`
  - `[::]`

- String methods

Updating list values: [], [:], [::]

- **Updating list values**: YES, they are mutable types, syntax is `L[index] = new_value`

`colors = ['red', 'green', 'blue', 'cyan']`

`colors[1] = 'yellow'` → same `colors` list, with modified value, `'yellow'`, for item in position 1

`colors[0] = None` → same `colors` list, with modified value and type, `None`, for item in position 0

`colors[0:3] = ('brown', 'magenta', 'pink')` → updating a subsequence of adjacent items

`colors[0:3:2] = ('brown', 'magenta')` → updating a subsequence of non-adjacent items

`colors[4] = 'purple'` → error! the list doesn't include an item at position 4 and the list cannot be *extended* in this way

`numbers = []` → defines an empty list, `numbers[0]` does not exist (yet)! List cannot be extended this way

`numbers[1]` → error! the list doesn't include an item at position 1 and the list cannot be *extended* in this way

Extending a list/tuple by adding multiple list elements: +, +=

- **Concatenation** operator + : add items from another list/tuple onto the end of the list

```
prime_numbers = [1, 3, 5, 7, 11]
other_primes = [13, 17, 19]
new_primes = prime_numbers + other_primes
```

→ *new* list with [1,3,5,7,11,13,17,19]

```
primes = [1, 3, 5, 7, 11, 13, 17]
primes = primes + [19, 23, 29]
```

→ `primes` has changed identity, it's a *new* list

```
primes = (1, 3, 5, 7, 11, 13, 17)
primes = primes + (19, 23, 29)
```

→ `primes` has changed identity, it's a *new* tuple

+ operator changes *identity, not in-place*

(check it with `print(id(primes))` before and after concatenation!)

Adding multiple list elements: +, +=

- Augmented notation for the + **operator addition**: +=

```
primes += [19, 23, 29]
```

same *high-level* result as

```
primes = primes + [19, 23, 29]
```

+= operator: `primes` doesn’t change identity, *in-place*

(check it with `print(id(primes))` before and after +=)

Duplication of lists/tuples: operator `*`, `*=`

- **Duplication Operator**: `*` creates multiple copies of an existing list/tuple

```
prime_numbers = [1, 3, 5]
repeat_primes = prime_numbers * 2
```
→ new list/tuple with [1,3,5, 1,3,5]

- Very useful to create lists/tuples where <u>all elements have the same value</u>

  `x = [1,1,1,1,1,1,1,1,1,1]` → list with 10 elements all initialized to integer value 1

  `x = [1]*10` → create a list with 10 elements all initialized to the integer value 1

➢ Augmented version of **operator multiplication**: `*=`

`a = a * b` *is the same in value as* `a *= b`

`*=` operator: `primes` doesn't change identity, *in-place*

Basic membership operators: `in`, `not in`

- Operator `in`: **Membership**, returns `True` if item belongs to the list/tuple, `False` otherwise

```
prime_numbers = [1, 3, 5, 7, 11]
is_prime = 5 in prime_numbers          → new bool variable with value True
```

- Operator `not in`: **Membership**, returns `False` if item belongs to the list/tuple, `True` otherwise

```
prime_numbers = [1, 3, 5, 7, 11]
is_prime = 5 not in prime_numbers          → new bool variable with value False
```

Test your knowledge

Write the function `operations(t, n)` that takes as input a tuple `t` and an integer, `n`. The function returns a list `L` with the following contents. `L` includes all the elements of `t` at the odd positions.
If the first element of `L` is a string, the function prints out the `n`-th character of that string, of it exists.
Otherwise it prints out "Short string!"

```
def operations(t, n):
    L = list(t[1::2])
    if type(L[0]) == str:
        if len(L[0]) >= n:
            print(L[0][n-1])
        else:
            print("Short string!")
    return L
```