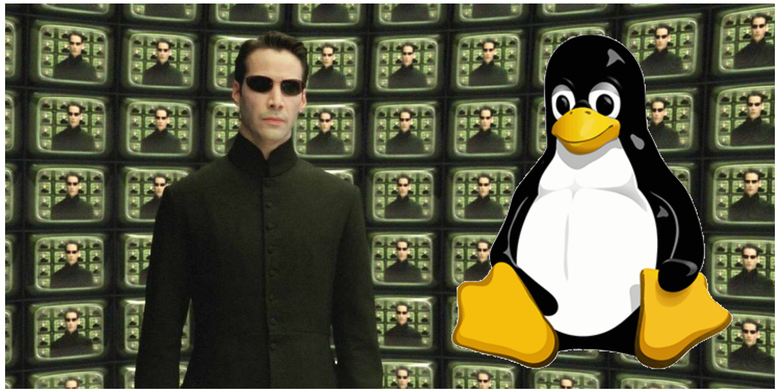
08.10.21 15:53

# CSCI 120
# Introduction to Computer Science and Programming I
# Lecture 4: Lists II



Jetic Gū

# Overview

- Focus: Basic Python Syntax
- Core Ideas:
  1. `tuple`
  2. Advanced `list` and `tuple`

# `tuple` **type data**

Immutable

P1
tuple

# Python `list`

- An **ordered** sequence of variables
  - `[1, 2, 3, 4, 5]`
    list instances are declared using square brackets, with values inside separated by comma
  - Indexed access
  - **Mutable: you can change the value of any elements inside a list using indexing**

Review

P1
tuple

# Python `tuple`

- An **immutable ordered** sequence of **immutable objects**
  - `x = 1, 2, 3, 4, 5`
    `y = ('a', 'b', 'c', 'd', 'e')`
    tuple instances are declared using parentheses, or just comma, with values inside separated by comma
  - Immutable objects: `int`, `float`, `str`, `tuples`
  - Indexed access
  - **Immutable: you can NOT change the value of any element inside a tuple**

Concept

P1
tuple

# Python `tuple`

```
a_list = [1, 2, 3, 4, 5]
for i in range(len(a_list)):
    a_list[i] = a_list[i] + 1  # This will work
```

`list` can be changed through indexing

```
a_tuple = (1, 2, 3, 4, 5)
for i in range(len(a_tuple)):
    a_tuple[i] = a_tuple[i] + 1  # This will NOT work
```

`tuple` can NOT be changed through indexing

- `tuple` can **NOT** have it's internal elements' values changed through indexing

Technical

# Python `tuple`

```
n, m = 12, 11
# Equivalent to n, m = (12, 11)
# Equivalent to
# n = 12
# m = 11
```

- `tuple` instances can be decomposed in value assignments
  - `list` cannot

# Python `tuple`

```
a = [(0, 0), (1, 2), (3, 4)]
for i, j in a:
    print("Coordinate: (", i, ",", j, ")")
```

- `tuple` instances can be decomposed in `for` loop value assignments
  - `list` cannot

# Python `tuple`

```
x_values = [0, 1, 2, 3, 4, 5, 6]
y_values = [0, 1, 4, 9, 16, 25, 36]
for i, j in zip(x_values, y_values):
    print("Coordinate: (", i, ",", j, ")")
```

- `zip(l1, l2, ...)` iterator
  - Takes multiple `list` objects `l1`, `l2`, `...` as parameters
  - Values are pairs of elements in `tuple`:
    `(l1[0], l2[0], ...)`,
    `(l1[1], l2[1], ...)`, ..., until the shortest list runs out of elements

# Similar stuff to `list`

```
a = (,)  # an empty tuple
b = ("red",)  # a tuple with one element
c = (1, 1.0, "funny stuff")
d = 1, 1.0
```

- A `tuple` instance can have different types of values, as long as they are all immutable

# Similar stuff to `list`

```
c = [1, 1.0, "funny stuff"]
c_tuple = tuple(c)
d_tuple = tuple(range(10))
```

- `list` objects can be converted to `tuple` using converter `tuple()`
- `tuple` can also come straight from `iterator`s, just like `list`

# Similar stuff to `list`

```
a = (1, 2, 3, 4, 5)
b = (5, 6, 7, 8, 9)
c = a + b  # c = (1, 2, 3, 4, 5, 5, 6, 7, 8, 9)
d = b + a  # d = (5, 6, 7, 8, 9, 1, 2, 3, 4, 5)
e = a[1:3]
f = (b + a)[4:8]
```

- Multiple `tuple` instances can be concatenated as one using the plus sign
- `tuple` instances also support slicing

# More advanced `list` and `tuple`

P2
Advanced stuff

# Creating empty `list` / `tuple`

```
x = (0,) * 5
# x = (0, 0, 0, 0, 0)

y = [0] * 5
# y = [0, 0, 0, 0, 0]
```

- Python `list` and `tuple` objects can be multiplied by an `int`, similar to how concatenation works
  - `[0] * 5` is equivalent to `[0] + [0] + [0] + [0] + [0]`

Technical

P2
Advanced stuff

# Creating a `list` using `for` loop

```
x = [i * i for i in range(10)]

# The above code is equivalent to
x = []
for i in range(10):
    x.append(i * i)
```

- You can place a for loop inside a square bracket to compose a list
  - Syntax: `new_list = [EXPRESSION for VAR in ITERATOR]`

Technical

P2
Advanced stuff

# Initialising an $n \times m$ matrix

```
x = [[0] * m for i in range(n)]

# Equivalent to
x = []
for i in range(n):
    x.append([0] * m)
```

- Create a nested list, use it as a 2D array (matrix) of numbers

Example

P2
Advanced stuff

# `list` using `for` loop and condition

```
x = [i for i in range(100) if i % 2 == 0]
# All even numbers under 100

# The above code is equivalent to
x = []
for i in range(100):
    if i % 2 == 0:
        x.append(i)
```

- You can place a for loop inside a square bracket to compose a list
  - Syntax:
    `new_list = [EXPRESSION for VAR in ITERATOR if EXPRESSION]`

Technical

P2
Advanced stuff

# Convert `str` of multiple `int`

```
x = "1 2 3 4 5 56 -10"
x = [int(i) for i in x.split()]
# Convert to list of integers

x = "1 2 3 4 5 56 -10"
x = [int(i) for i in x.split() if int(i) > 0]
# Convert to list of integers, but only nonnegative ones
```

- Efficient handling of `str` input of numbers

Example

P2
Advanced stuff

# Vector addition

```
vec_1 = [0, 1, 3, 5, 9]
vec_2 = [5, 7, 9, 3, 3]

sum = [i + j for i, j in zip(vec_1, vec_2)]
```

- Vector sum: element-wise addition of two lists

Example