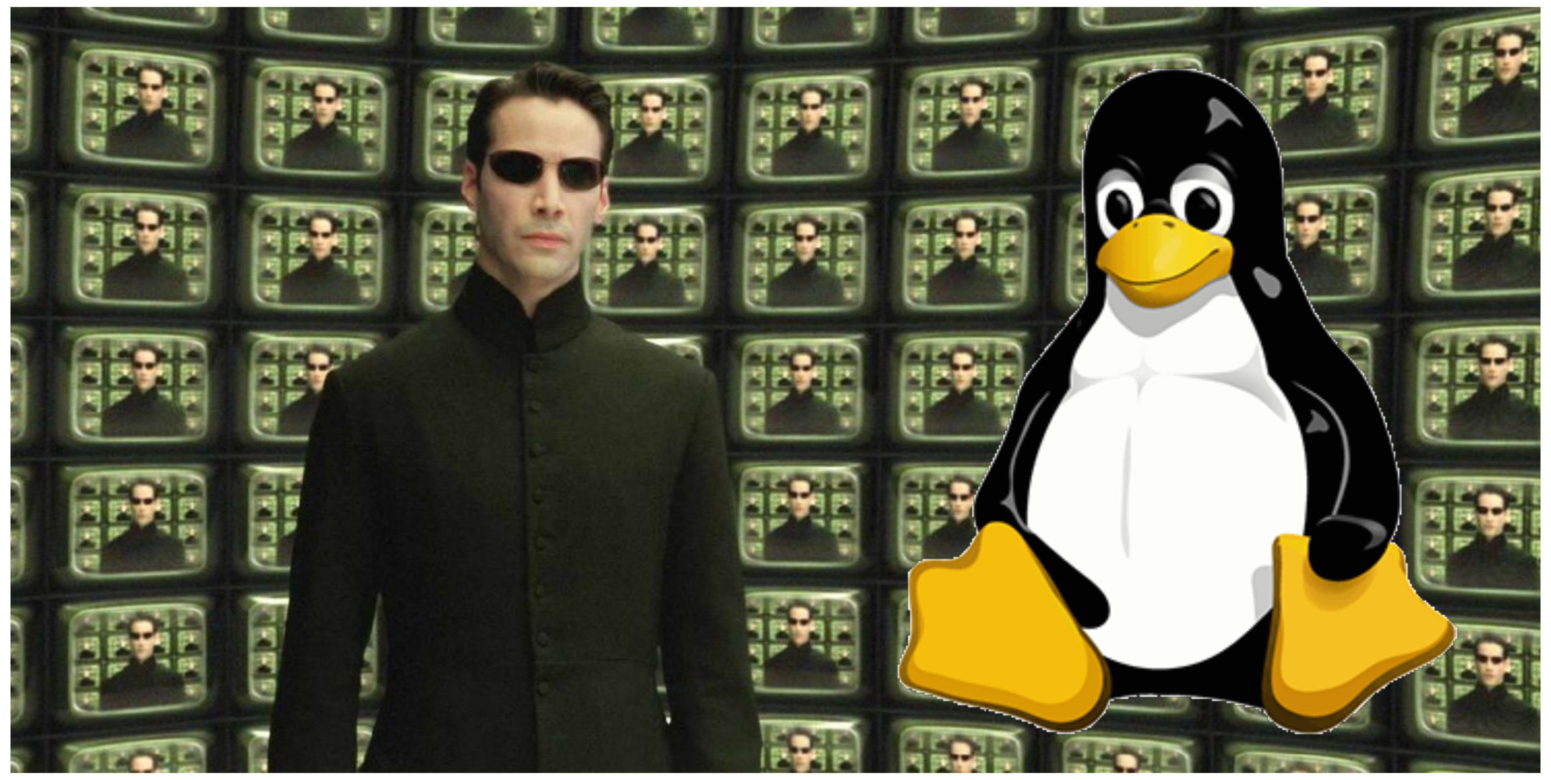
21.09.21 15:52

CSCI 120
Introduction to Computer Science and Programming I
Lecture 2: Loops I



Jetic Gū

Overview

- Focus: Basic Python Syntax
- Core Ideas:
  1. for loops, iterators
  2. Tips

for loops

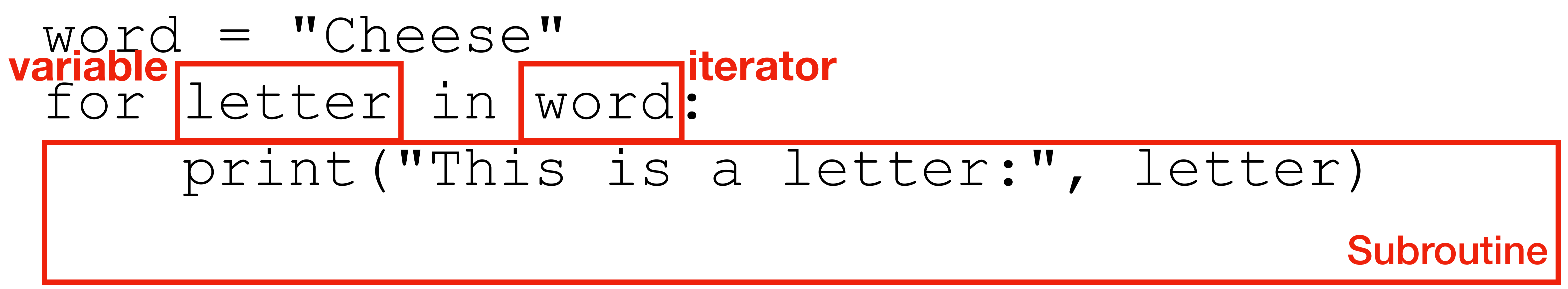
Doing things, over and over again

Why loops

- Imagine
  - Having to print the same message to `stdout` 1000 times
  - Having to go through an entire array (or string) to process each element (or character)
  - Keep processing input as they arrive (endless repeat)

Concept

Looping through a string

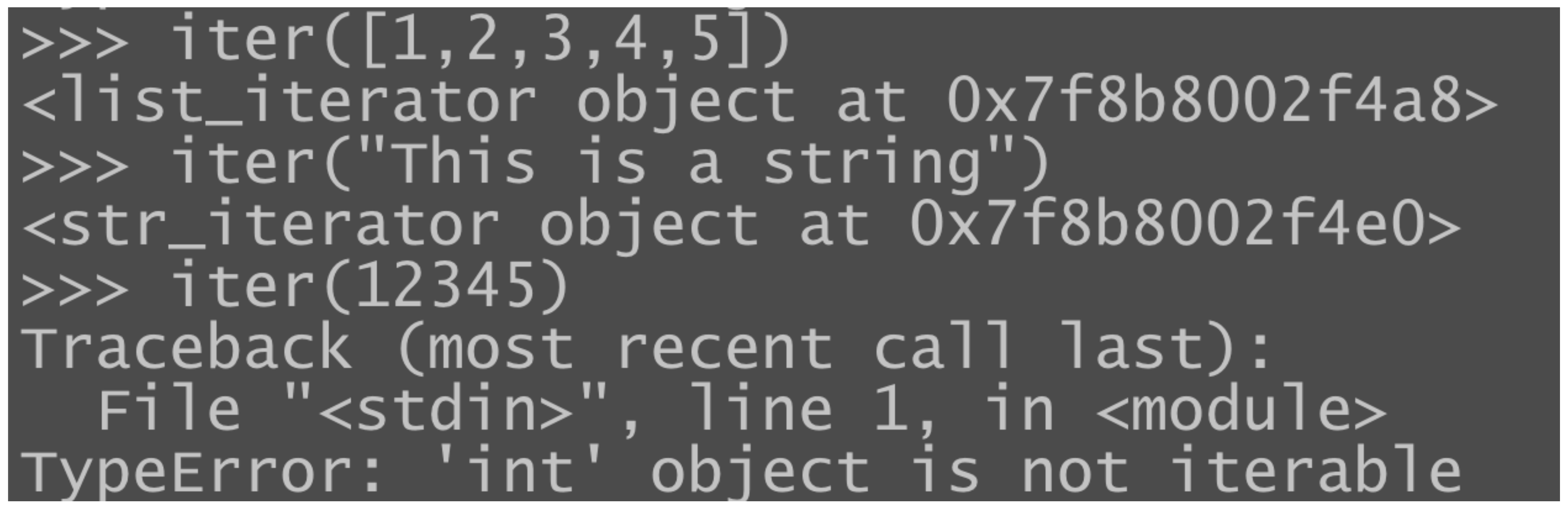


```
word = "Cheese"
for letter in word:
    print("This is a letter:", letter)
```

- For loop
  - 3 parts: a new **variable**, the **iterator**, and the **subroutine**
  - In this case, variable `letter` is going to take on values `"C"`, `"h"`, `"e"`, `"e"`, `"s"`, `"e"`, and for every value, the subroutine will be executed

Python Iterator

```
for VARIABLE in ITERATOR:
    SUBROUTINE...
```



```
>>> iter([1,2,3,4,5])
<list_iterator object at 0x7f8b8002f4a8>
>>> iter("This is a string")
<str_iterator object at 0x7f8b8002f4e0>
>>> iter(12345)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

- Python data types can be divided into **iterable** ones, and **non-iterable** ones
  - Iterable data types: can be converted into iterators using `iter()` function
    - `list`s: e.g. `[1, 2, 3, 4, 5]`
    - `str`: e.g. `"This is a string"`

Iterating through a list of numbers

```
sum = 0
for i in [1,2,3,4,5]:
    sum = sum + i
print("The sum from 1 to 5 is:", sum)
```

- Iterating through a list of values to calculate the sum
  - What if it is up to 100? or 1000?

The `range()` function

```
sum = 0
for i in range(101):
    sum = sum + i
print("The sum from 0 to 100 is:", sum)
```

- `range(101)`
  Generates an iterator, with values from 0, 1, ..., 100
- `range(var)   # var must be integer`
  Generates an iterator, with values from 0, 1, ..., `var-1`
- Can be converted to a list using `list()`
  `list(range(10))` will give you `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

Concept

The `range()` function

- Options for the `range()` function
  - `range(var)   # var must be integer`
    Generates an iterator, with values from 0, 1, ..., `var-1`
  - `range(i, j)   # i and j must be integer`
    Generates an iterator, with values from `i`, `i+1`, ..., `j-1`
    - `list(range(-2,2))` will give you `[-2, -1, 0, 1]`
  - `range(i, j, inc)   # i and j and inc must be integer`
    Generates an iterator, with values from `i`, `i+inc`, `i+2 * inc` ..., `j-1`
    - `list(range(1,9,3))` will give you `[1, 4, 7]`
    - `list(range(6,1,-1))` will give you `[6, 5, 4, 3, 2]`

Some tips for `for`

New stuff included.

Tip 0: Multiple test cases

```
n = int(input())
for i in range(n):
    # do stuff...
```

- In algorithm contests as well as on OJ, it is not uncommon for you to encounter multiple test cases
  - You can use `for` to solve them

Technical

Tip 1: Layered `for` loops

```
n = int(input())
for i in range(n):
    for j in range(n):
        print("(", i, ",", j, ")", end="\t", sep="")
    print("")
```

- `for` statements inside another layer of loop
  - The code above will print a nice matrix of $n \times n$ size, each position contains its coordinate

Tip 2: Break Prematurely

```
target = int(input())
for i in range(1000):
    print(i)
    if i == target:
        break
print("for loop terminated")
```

- `break` statement
  - When certain condition is met, you might want to end a for loop prematurely
  - `break` will terminate the most immediate layer of loop

Tip 3: Looping the indices

```
a = [1,2,3,4,5]
b = [6,7,8,9,10]
dot = 0
for i in range(len(a)):
    dot = a[i] * b[i]
```

- Instead of iterating through an entire list/string, you may also generate an iterator of indices, so you can access elements by their indices
- In this example, you are calculating the dot product for two vectors `a` and `b`

Technical