CDS 230
Modeling and Simulation I

Module 5

Loops

Dr. Hamdi Kavak
http://www.hamdikavak.com
hkavak@gmu.edu

GEORGE MASON UNIVERSITY

Center for Social Complexity

This lecture

We will combine our knowledge on all the topics covered so far

- Numbers, variables
- Mathematical statements
- If, else, etc. (i.e., control flow)
- Strings
- Collections

Loops and iteration

- Imagine a code piece should be executed multiple times.
  - E.g.: Simulate the projection of a free falling object every 0.1 seconds.
- Without loops, your solution would look like this ========>
- Thanks to loops and iteration, we don't need to follow that path.

```
y1=50 # initial position
vy=0 # initial velocity
g=9.8 # gravity

t=0.0
new_position = y1+vy*t-0.5*g*t**2
print(f"At time={t}, the position of the object is at {new_position}")

t=0.1
new_position = y1+vy*t-0.5*g*t**2
print(f"At time={t}, the position of the object is at {new_position}")

t=0.2
new_position = y1+vy*t-0.5*g*t**2
print(f"At time={t}, the position of the object is at {new_position}")

t=0.3
new_position = y1+vy*t-0.5*g*t**2
print(f"At time={t}, the position of the object is at {new_position}")
```

```
At time=0.0, the position of the object is at 50.0
At time=0.1, the position of the object is at 49.951
At time=0.2, the position of the object is at 49.804
At time=0.3, the position of the object is at 49.559
```

The `for` loop

- Allows us executing a code block while iterating over “iterable” objects (`lists`, `tuples`, and `strings`).
  - Iterable means **ordered** sequence of items.

- Syntax:
```
for item in iterable_object:
    # code block
```

- Indentation is same as the `if` statement.

Code example: `for` loop

[1]:
```
cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
for city in cities:
    print(city)
```

```
Fairfax
Alexandria
Reston
Herndon
Vienna
Oakton
Centreville
```

[2]:
```
for letter in "GMU":
    print(letter)
```

```
G
M
U
```

[3]:
```
coordinates = [(0,0), (0.4, 0.1), (0.7, 0.2), (0.99, 0.3)]
for coor in coordinates:
    print("x,y: ",coor)
```

```
x,y:  (0, 0)
x,y:  (0.4, 0.1)
x,y:  (0.7, 0.2)
x,y:  (0.99, 0.3)
```

Code example: nested `for` loop

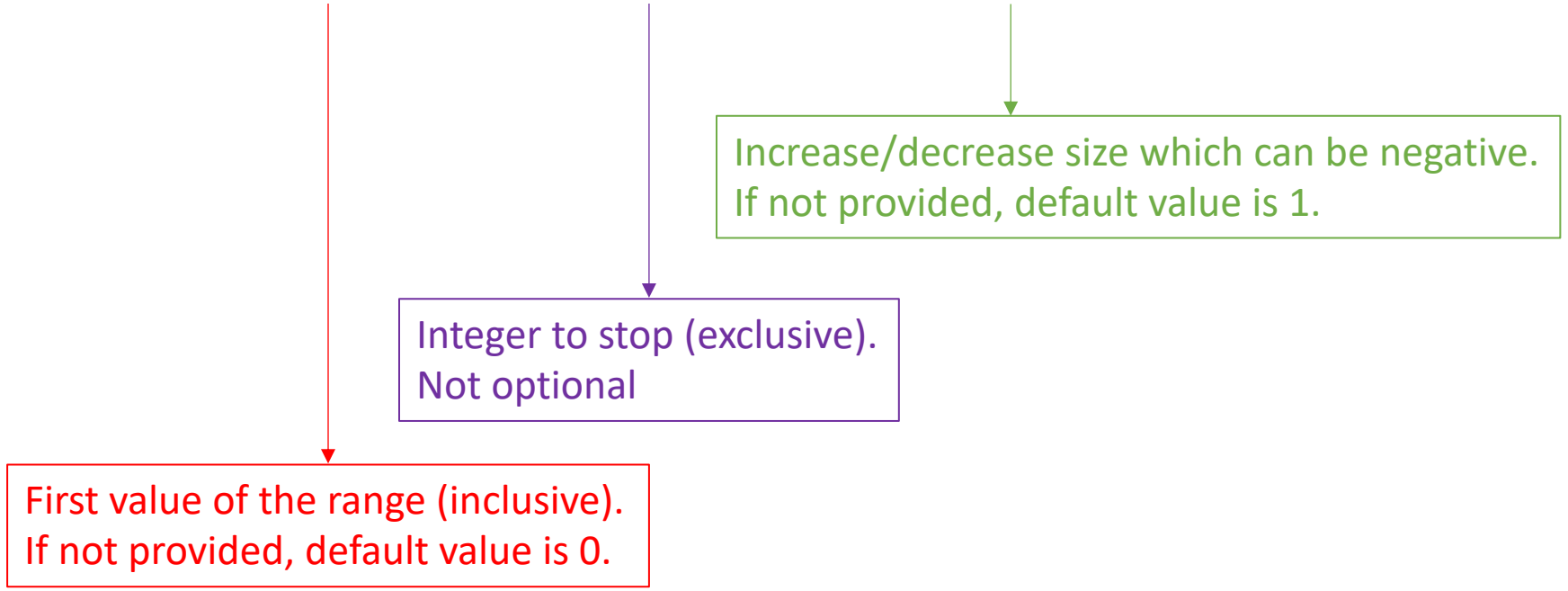
```
[4]: cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
for city in cities:
    for letter in city:
        print(letter,end=".")
    print()
```

```
F.a.i.r.f.a.x.
A.l.e.x.a.n.d.r.i.a.
R.e.s.t.o.n.
H.e.r.n.d.o.n.
V.i.e.n.n.a.
O.a.k.t.o.n.
C.e.n.t.r.e.v.i.l.l.e.
```

`print(end=".")` => prints a dot as the last character (not \n)

The `range` type

- Creates iterable **integers on demand** based on given parameters.
- Does **not** create a list!
- Syntax: `range(start, end, stride)`



Code example: `range`

- `range(6)` 0, 1, 2, 3, 4, 5
- `range(1,5)` 1, 2, 3, 4
- `range(0,10,2)` 0, 2, 4, 6, 8
- `range(9,0,-2)` 9, 7, 5, 3, 1

```
for num in range(9, 0, -2):
    print(num)
```

```
9
7
5
3
1
```

```
for num in range(5):
    print(num*num)
```

?

```
0
1
4
9
16
```

enumerate

- Produces index values while iterating over a list or tuple.
  - E.g.,: You have the city list and want to print them with a counter:

```
[1]: cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
```

1 . Fairfax
2 . Alexandria…

- Two ways to achieve the desired output

```
for index in range(len(cities)):
    print(index+1,".",cities[index])
```

```
1 . Fairfax
2 . Alexandria
3 . Reston
4 . Herndon
5 . Vienna
6 . Oakton
7 . Centreville
```

```
for index, city in enumerate(cities):
    print(index+1,".",city)
```

```
1 . Fairfax
2 . Alexandria
3 . Reston
4 . Herndon
5 . Vienna
6 . Oakton
7 . Centreville
```

zip

- Used when two or more sequences are iterated at the same time.
- Syntax: `zip(first_iterable, second_iterable,…)`

```
indexes = [1, 2, 3, 4, 5, 6, 7]
cities = ["Fairfax", "Alexandria", "Reston", "Herndon", "Vienna", "Oakton", "Centreville"]
```

```
for item in zip(indexes, cities):
    print(item)
```

```
(1, 'Fairfax')
(2, 'Alexandria')
(3, 'Reston')
(4, 'Herndon')
(5, 'Vienna')
(6, 'Oakton')
(7, 'Centreville')
```

```
for item in zip(indexes, cities):
    print(item[0],".",item[1])
```

```
1 . Fairfax
2 . Alexandria
3 . Reston
4 . Herndon
5 . Vienna
6 . Oakton
7 . Centreville
```

Code example: `zip`

More than two iterables

```
list1 = [0,1,2]
list2 = ["zero", "one", "two"]
list3 = ["cero", "uno", "dos"]
for item in zip(list1, list2, list3):
    print(item)
```

```
(0, 'zero', 'cero')
(1, 'one', 'uno')
(2, 'two', 'dos')
```

Iterables with different lengths?

```
list1 = [0,1,2]
list2 = ["zero", "one", "two", "three"]
for item in zip(list1, list2):
    print(item)
```

```
(0, 'zero')
(1, 'one')
(2, 'two')
```

```
list1 = [0,1,2, 3]
list2 = ["zero", "one", "two"]
for item in zip(list1, list2):
    print(item)
```

```
(0, 'zero')
(1, 'one')
(2, 'two')
```

The `while` loop

- Executes a block of code as long as a given condition holds
- Syntax:
```
while condition is true:
    # code block
```
- Indentation is same as in the `if` statement
- More flexible than the `for` loop but have the potential to run into an "infinite loop" problem

Code example: `while` loop

Use as counter

```
index = 0
while index < 5:
    index = index + 1 # or index+= 1
    print(index)
```

```
1
2
3
4
5
```

Iterate over lists

```
list2 = ["zero", "one", "two"]
index = 0
while index < len(list2):
    print(list2[index])
    index+= 1
```

```
zero
one
two
```

Control flow in loops

- We can insert additional control flow statements within loops
- You already learned `if` … `else` …. `elif` statements

```
for num in range(11):
    if num % 2 == 0:
        print(num, "is an even number.")
    else:
        print(num, "is an odd number.")
```

```
0 is an even number.
1 is an odd number.
2 is an even number.
3 is an odd number.
4 is an even number.
5 is an odd number.
6 is an even number.
7 is an odd number.
8 is an even number.
9 is an odd number.
10 is an even number.
```

- Other control statements:
  - `break`: immediately terminates the loop
  - `continue`: forces the next iteration without executing the rest of the iteration
  - `pass`: does nothing (placeholder)

Example code: `break` and `continue`

```
num = 0
max_number = 5
while True:
    if num > max_number:
        break

    if num % 2 == 0:
        print(num, "is an even number.")
    else:
        print(num, "is an odd number.")

    num += 1
```

```
0 is an even number.
1 is an odd number.
2 is an even number.
3 is an odd number.
4 is an even number.
5 is an odd number.
```

```
for num in range(11):
    if num % 2 == 0:
        continue
    print(num, "is an odd number.")
```

```
1 is an odd number.
3 is an odd number.
5 is an odd number.
7 is an odd number.
9 is an odd number.
```

any and all

- any() : tests whether any items in an iterable object is True.
- all() : tests whether all items in an iterable object is True.
- If an iterable item is not bool type, it will be converted to bool.

```
first_list = ["Hello", "World", 2000, True]
empyt_tuple = tuple()
one_elem_list = list([True])
print( any(first_list))
print( all(first_list))
print( any(empyt_tuple))
print( all(empyt_tuple))
```

The last item in the list is True, so the result is True

Each item will be converted to bool if they are not already the Boolean type, so the result is True

Empty iterable returns False when passed to any().

Empty iterable returns True when passed to all().

In Python, there's more than one way to skin a cat

- Your task: generate values [0.0, 0.1,…,3.0]

```
time = []
for i in range(31):
    time.append(i/10.0)
print (time)
```

```
time = []
i = 0
while i <= 30:
    time.append(i/10)
    i += 1
print (time)
```

```
time = [i/10 for i in range(31)]
print(time)
```

Print example

- You are given two variables: `num_of_character = 10` and `charachter_to_print = "*”`. Write a program that prints the character for given number of times in the same line using `for` or `while` loops.
- Based on the above parameters, your program should print 10 stars “**********”.
- You need to write the code in a way that if you change the value of the above variables, it should be reflected in the output.
  - Given `num_of_character = 3`, `charachter_to_print = ”-”`, your program should print three dashes in the same line “---”.
- `print(end=“.”)` => prints a dot as the last character (not \n)

Example problem

- Given an object dropped from a height of $100\ m$ in a frictionless environment under gravity's influence. Write Python code that computes the falling object's new position every 0.1 seconds until it reaches the ground (y = 0) (assume no bouncing). Keep track of the distance (using a `list`) between the current location and the next predicted location every 0.1 seconds and print the result.

$$y_2 = y_1 + v_y\ t - \frac{1}{2} g\ t^2$$

Starting height = y_1.
Ending height = y_2 .
Initial velocity = v_y .
Gravity = $g = 9.8$
Time = t