FOP

Repeating Code

2025-26

WESTMINSTER
International University in Tashkent

In this lecture …

- How can we repeat instructions with little effort?
- What is a loop?
- How can a range of integers be created?
- How can a list of integers be created?
- How can a loop be interrupted?

Repeating Code

- Imagine we want apply the same instruction to a list of floats (e.g. print each value raised to the power of two)
- If the list has 1000 floats, we'd need to write the instruction 1000 times:

```
print(a_list[0]**2)
print(a_list[1]**2)
…
print(a_list[999]**2)
```

- The `for` loop solves that problem

for

- Automatically iterates over a sequence of values
- Syntax:

```
for a_variable in an_iterable:
    instructions
```

- Example:

```
for c in 'Hello':
    print(c)
```

1 minute problem

- Create a program to print each element in the following list:

```
a_list = [1.5, 2, 2.5,
 math.sqrt(2)]
```

- Solution:

```
for value in a_list:
  print(value)
```

```
1.5
2
```

Another 1 minute problem

- Create a program that prints only the upper case characters of a string given by the user
- Example:

```
Please, insert text: Hello I am
 António


H
I
A
```

Solution

```
text = input('Please, insert text: ')

for c in text:
    if c.isupper():
        print(c)
```

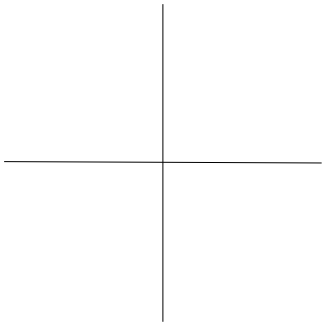
Iterating over a range of numbers

- In order to create a range of numbers we need a new type
  - `range`
- `range(stop)`
  - Creates an integer range of values starting in 0 up to `stop-1`
- `range(start, stop)`
  - Creates an integer range of values starting in `start` up to `stop-1`

Examples

```
for val in range(5):
    print(val)
```

```
for i in range(1, 6):
    print(i)
```



```
for i in range(-5, 5):
    print(i)
```

```
for i in range(4):
    print(i**2)
```

Creating lists of integers

- Easy! Wrap the `range` with a `list` constructor
- Examples:

```
a_range = range(5)
 a_list = list(a_range)
 print(a_list)
 [0, 1, 2, 3, 4]

another_list = list(range(-5, 6))
 print(another_list)
 [-5, -4, -3, -2, -1, 0, 1, 2, 3,
 4, 5]
```

Ranges as indices

- Iterating a list using indices

```
a_list = ['a', 'b', 'c']
for i in range(3):
  print(a_list[i])
```

Instead of:

```
for c in a_list:
      print(c)
```

- Even more clever

```
a_list = ['a', 'b', 'c']
for i in range(len(a_list)):
  print(a_list[i])
```

2 minute problem

- Create a program that changes the contents of the following list:

```
a_list = [2.5, 2, 4, 1, 3.3]
```

- in a way that each position will have the double of its current value

Solution

```
a_list = [2.5, 2, 4, 1, 3.3]
for i in range(len(a_list)):
  a_list[i] = 2 * a_list[i]
```

- Checking the new content

```
print('The new content is:')
for i in range(len(a_list)):
  print(a_list[i])
```

Nested Loops

- Loops inside loops (wow!)
- One loop inside the other, means:
  - Execute **the inner loop iterations** once for each iteration of the outer loop
- Example:

```
for i in range(2):
    for j in range(3):
        print('Loop:', i, j)
```

Two minute exercise
A program that outputs:

1 times table
1 × 1 = 1
1 × 2 = 2
1 × 3 = 3
1 × 4 = 4
1 × 5 = 5
1 × 6 = 6
1 × 7 = 7
1 × 8 = 8
1 × 9 = 9
1 × 10 = 10

2 times table

$2 \times 1 = 2$

$2 \times 2 = 4$

$2 \times 3 = 6$

$2 \times 4 = 8$

$2 \times 5 = 10$

$2 \times 6 = 12$

$2 \times 7 = 14$

$2 \times 8 = 16$

$2 \times 9 = 18$

$2 \times 10 = 20$

Solution

```
for i in range(1,3):
    print(i, 'times table')
    for j in range(1, 11):
        print(i, '×', j, '=', i * j)
    print()
```

Iterating over Lists of Lists

```
data = [['Antonio', 41], ['Maria', 61, 'PT']]

for row in data:
    for item in row:
        print(item, '', end='')
    print()
```

2 minute problem

- Given the following list:
- `data = [['Antonio', 41], ['Maria', 61, 'PT']]`
- Create a program that, using indexed `for`s, produces the following output:

```
Antonio 41
 Maria 61 PT
```

Solution

```
data = [['Antonio', 41], ['Maria', 61, 'PT']]

for i in range(len(data)):
    for j in range(len(data[i])):
        print(data[i][j], ' ', end='')
    print()
```

Repeating while something is `True`

- `for` loops are usually used when the number of iterations is know in advance
- `while` loops can be used when we don't know how many times we want to repeat something
- While loops repeat while a condition is `True`
- Syntax:

```
while condition:
    instructions
```

Examples

```
value = 10
while value >=0:
     print(value)
     value -= 1 # same as: value = value - 1
```

```
value = -1
while value < 0:
     value = int(input('Insert a number >= 0: '))
print('You inserted', value)
```

Breaking a loop

- `for` and `while` loops are supposed to run all the statements in their body
- `break` allows us to immediately interrupt a loop
- The program resumes after the loop
- Example:

```
for k in range(10):
  print(k)
  if k == 5:
        break
```

Breaking a `while`

- Before we had:

```
value = -1
while value < 0:
    value = int(input('Insert a number >= 0: '))
print('You inserted', value)
```

- Now we can have:

```
value = 0
while True:
    value = int(input('Insert a number >= 0: '))
    if value >= 0:
        break
print('You inserted', value)
```

1 minute problem

- Write a program that prints the index of the first upper case in an string, or -1 if there is no upper case

Solutions

- Would this be OK?

```
text = input()
index = -1

for k in range(len(text)):
      if
text[k].isupper():
                    index = k

print('Index:', index)
print('Bye!')
```

- Consider with the text:

```
'Hello People!'
```

- What about this?

```
text = input()
index = -1

for k in range(len(text)):
      if
text[k].isupper():
              index = k
              break

print('Index:', index)
print('Bye!')
```

Skipping an iteration

- It is possible to skip one (or more) iteration of the loop, without completely breaking it
- `continue` allows that
- The program won't execute the remaining instructions in the loop block and jumps to the next iteration
- Example:

```
for k in range(10):
        if k == 5:
            continue
    print(k)
```

2 minute problem

- Let's say I have a problem with number 5
- I want a program that prints the multiplication table but, every time there is digit 5 in any position of the operation, it won't print it
- For example, it will print 1 × 1 = 1, 1 × 2 = 2, but it won't print 2 × **5** = 10, 6 × 9 = **5**4

Solution

```
for i in range(1, 11):
    if i == 5:
        continue

    for j in range(1, 11):
        if j == 5 or '5' in str(i*j):
            continue

        print(i, '×', j, '=', i*j)

  print()
```

Advice

- Try to avoid `break` and `continue`
- Makes code harder to read
- Use it only if otherwise the program will become too complex

In this lecture …

- How can we repeat instructions with little effort?
- What is a loop?
- How can a range of integers be created?
- How can a list of integers be created?
- How can a loop be interrupted?

Further reading

- PP, chapter 9