Programming Essentials in Python

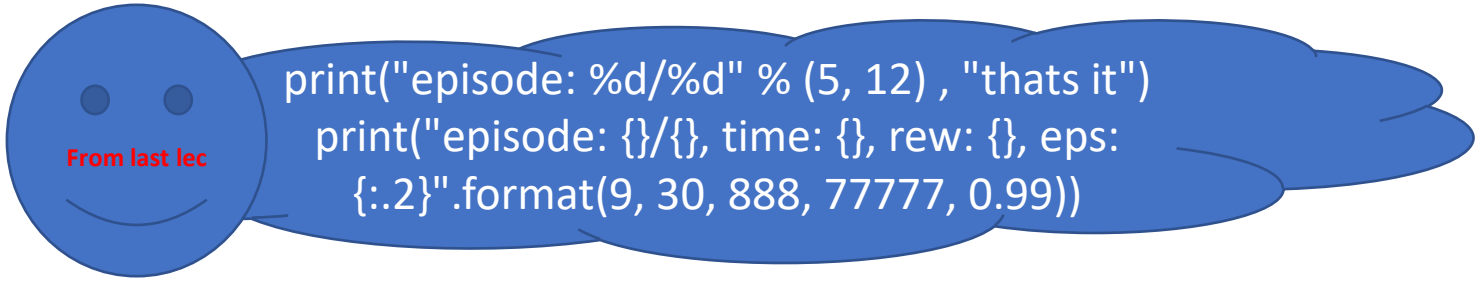
Introduction to Python

Dhulfiqar A Alwahab **Dr.Laki Sandor**

What you will learn

- **Pseudocode and introduction to loops**
- **Loops in Python**
- **Break and continue**
- **Logic and bit operations**
- **All about list in Python**

Introduction to Python



Pseudocode

Pseudocode is an algorithm that you write before you start programming in any languages (i.e flowchart) and executing
A piece of code more than one is called a **loop**.

Python comes with a lot of **built-in** functions that will do
The work for you. See the box in the right! And use min().

Looping your code with while

while conditional_expression:
4spaces or tab instruction

```
while conditional_expression:
    instruction_one
    instruction_two
    instruction_three
    :
    :
    instruction_n
```

```
# read three numbers
number1 = int(input("Enter the first number: "))
number2 = int(input("Enter the second number: "))
number3 = int(input("Enter the third number: "))

# check which one of the numbers is the greatest
# and pass it to the largest_number variable

largest_number = max(number1, number2, number3)

# print the result
print("The largest number is:", largest_number)
```

Introduction to Python

- **An infinite loop (endless loop)**

```
while True:
    print("I'm stuck inside a loop.")
```

- Simplify the code

while number !=0 -> while number:

if number % 2 == 1: -> if number % 2:

- See the difference

```
counter = 5
while counter != 0:
    print("Inside the loop.", counter)
    counter -= 1
print("Outside the loop.", counter)
```

```
counter = 5
while counter:
    print("Inside the loop.", counter)
    counter -= 1
print("Outside the loop.", counter)
```

```
#    Analyze  the following code
#A program that reads a sequence of numbers
# and counts how many numbers are even and how many are odd.
# The program terminates when zero is entered.

odd_numbers = 0
even_numbers = 0

# read the first number
number = int(input("Enter a number or type 0 to stop: "))

# 0 terminates execution
while number != 0:
    # check if the number is odd
    if number % 2 == 1:
        # increase the odd_numbers counter
        odd_numbers += 1
    else:
        # increase the even_numbers counter
        even_numbers += 1
    # read the next number
    number = int(input("Enter a number or type 0 to stop: "))

# print results
print("Odd numbers count:", odd_numbers)
print("Even numbers count:", even_numbers)
```

Introduction to Python

Lab 2.1 (3 mins)

- Scenario
- A junior magician has picked a secret number. He has hidden it in a variable named secret_number. He wants everyone who run his program to play the Guess the secret number game, and guess what number he has picked for them. Those who don't guess the number will be stuck in an endless loop forever! Unfortunately, he does not know how to complete the code.
- Your task is to help the magician complete the code in the editor in such a way so that the code:
- will ask the user to enter an integer number;
- will use a while loop;
- will check whether the number entered by the user is the same as the number picked by the magician. If the number chosen by the user is different than the magician's secret number, the user should see the message "Ha ha! You're stuck in my loop!" and be prompted to enter a number again. If the number entered by the user matches the number picked by the magician, the number should be printed to the screen, and the magician should say the following words: "Well done, muggle! You are free now."
- The magician is counting on you! Don't disappoint him.

```
# note : look how triple quotes for multi-line ( printing and commenting )
secret_number = 777

print(
"""
+================================+
| Welcome to my game, muggle!    |
| Enter an integer number        |
| and guess what number I've     |
| picked for you.                |
| So, what is the secret number? |
+================================+
""")
```

Introduction to Python

- **Looping your code with for**

For is not the same as **while** because sometimes it's more important **to count the "turns" of the loop** than to check the conditions. Check the two same code below

```
i = 0
while i < 100:
    # do_something()
    i += 1
```

```
for i in range(100):
    # do_something()
    pass
```

Note : The range() function invocation may be equipped with two arguments, not just one:
in this example-1 the first value will be 2 and the last value will be **7** not 8. Example-2 will print 2 and 5

Example -1

```
for i in range(2, 8):
    print("The value of i is currently", i)
```

Example -2

```
for i in range(2, 8, 3):  # the third one is called increment (default is 1)
    print("The value of i is currently", i)
```

- **What is the output ?**

```
for i in range(1, 1):
    print("The value of i is currently", i)
```

```
for i in range(2, 1):
    print("The value of i is currently", i)
```

What is the starting value of exp?

```
pow = 1
for exp in range(16):
    print("2 to the power of", exp, "is", pow)
    pow *= 2
```

Introduction to Python

Lab

- **Scenario**
- Do you know what Mississippi is? Well, it's the name of one of the states and rivers in the United States. The Mississippi River is about 2,340 miles long, which makes it the second longest river in the United States (the longest being the Missouri River). It's so long that a single drop of water needs 90 days to travel its entire length!
- The word Mississippi is also used for a slightly different purpose: to count mississippily.
- If you're not familiar with the phrase, we're here to explain to you what it means: it's used to count seconds.
- The idea behind it is that adding the word Mississippi to a number when counting seconds aloud makes them sound closer to clock-time, and therefore "one Mississippi, two Mississippi, three Mississippi" will take approximately an actual three seconds of time! It's often used by children playing hide-and-seek to make sure the seeker does an honest count.
- our task is very simple here: write a program that uses a for loop to "count mississippily" to five. Having counted to five, the program should print to the screen the final message **"Ready or not, here I come!"**
- Use the skeleton we've provided in the editor.
- EXTRA INFO
- Note that the code in the editor contains two elements which may not be fully clear to you at this moment: the import time statement, and the sleep() method. We're going to talk about them soon.
- For the time being, we'd just like you to know that we've imported the time module and used the sleep() method to suspend the execution of each subsequent print() function inside the for loop for one second, so that the message outputted to the console resembles an actual counting. Don't worry - you'll soon learn more about modules and methods.

```
import time

# Write a for loop that counts to five.
   # Body of the loop - print the loop iteration number
# and the word "Mississippi".
    # Body of the loop - use: time.sleep(1)

# Write a print function with the final message.
```

Expected output

1 Mississippi
2 Mississippi
3 Mississippi
4 Mississippi
5 Mississippi

Introduction to Python

- The break and continue statements

1- To refrain from further execution of the loop's body and go further;

2- to start the next turn of the loop without completing the execution of the current turn.

break - exits the loop immediately, and unconditionally ends the loop's operation; the program begins to execute the nearest instruction after the loop's body;

continue - behaves as if the program has suddenly reached the end of the body; the next turn is started and the condition expression is tested immediately

Run and test (3 min)

```
largestNumber = -99999999
counter = 0

while True:
    number = int(input("Enter a number or type -1 to end program: "))
    if number == -1:
        break
    counter += 1
    if number > largestNumber:
        largestNumber = number

if counter != 0:
    print("The largest number is", largestNumber)
else:
    print("You haven't entered any number.")
```

Run and test (3 min)

```
largestNumber = -99999999
counter = 0

number = int(input("Enter a number or type -1 to end program: "))

while number != -1:
    if number == -1:
        continue
    counter += 1

    if number > largestNumber:
        largestNumber = number
    number = int(input("Enter a number or type -1 to end program: "))

if counter:
    print("The largest number is", largestNumber)
else:
    print("You haven't entered any number.")
```

Introduction to Python

- **The while loop and the else branch** The else in the loop branch is always executed once, regardless of whether the loop has entered its body or not.

```
i = 1
while i < 5:
    print(i)
    i += 1
else:
    print("else:", i)

OUTPUT
1
2
3
4
Else: 5
```

- **The for loop and the else branch**

```
What will be the OUTPUT?

i = 111
for i in range(2, 1):
    print(i)
else:
    print("else:", i)
```

```
for i in range(5):
    print(i)
else:
    print("else:", i)

OUTPUT
1
2
3
4
 else : 4
```

Introduction to Python

LAB -2.2-

- Scenario
- In 1937, a German mathematician named Lothar Collatz formulated an intriguing hypothesis (it still remains unproven) which can be described in the following way:
- take any non-negative and non-zero integer number and name it c0;
- if it's even, evaluate a new c0 as c0 ÷ 2;
- otherwise, if it's odd, evaluate a new c0 as 3 × c0 + 1;
- if c0 ≠ 1, skip to point 2.
- The hypothesis says that regardless of the initial value of c0, it will always go to 1.
- Of course, it's an extremely complex task to use a computer in order to prove the hypothesis for any natural number (it may even require artificial intelligence), but you can use Python to check some individual numbers. Maybe you'll even find the one which would disprove the hypothesis.
- Write a program which reads one natural number and executes the above steps as long as c0 remains different from 1. We also want you to count the steps needed to achieve the goal. Your code should output all the intermediate values of c0, too.
- Hint: the most important part of the problem is how to transform Collatz's idea into a while loop - this is the key to success.
- Test your code using the data we've provided.

Test Data

Sample input: 15
Expected output:
46
23
70
35
106
53
160
80
40
20
10
5
16
8
4
2
1
steps = 17

Sample input: 16
Expected output:
8
4
2
1
steps = 4

Introduction to Python

Check-point !

In bitwise each number is represented by 32 bit

bitwise negation

Arg	~Arg
0	1
1	0

Bitwise operations (&, |, and ^)				
Arg A	Arg B	Arg B & Arg B	Arg A | Arg B	Arg A ^ Arg B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

- What is the output ?

n = 3
while n > 0:
  print(n + 1)
  n -= 1
else:
  print(n)
	for i in range(0, 6, 3):
	  print(i)

n = range(4)
 	
for num in n:
  print(num - 1)
else:
  print(num)
	Computer logic
	A and B
	A or B
	not Argument

x = 1 y = 0 z = ((x == y) and (x == y)) or not(x == y) print(not(z))	x = 4 y = 1 a = x & y b = x | y c = ~x d = x ^ 5 e = x >> 2 f = x << 2 print(a, b, c, d, e, f)

- **Logical vs. bit operations** i = 15 j = 22

Logical	Output	Bitwise	Output
I and j	True	I & j	00000000000000000000000000000110
not i	False	~i	11111111111111111111111111110000
I or j	True	i | j	

Abbreviated form

x = x & y	x &= y
x = x | y	x |= y
x = x ^ y	x ^= y

How do we deal with single bits

flagRegister = 0x1234 your bit is the third

Check the state of your bit

```
if flagRegister & theMask:  # in this ex Mask=8
   # my bit is set
else:
   # my bit is reset
```

Reset your bit

11111111111111111111111111110111

Binary left shift and binary right shift

The shift operators in Python are a pair of digraphs: << and >>

The left argument of these operators is an integer value whose bits are shifted. The right argument determines the size of the shift.

```
var = 10
varRight = var >> 1
varLeft = var << 2
print(var, varLeft, varRight)
          #  OUTPUT
          # 10 40 5
```

Introduction to Python

- **Lists.**
  - To declare a variable that could store more than one value

```
numbers = [10, 5, 7, 2, 1]
```

  - The elements inside a list may have different types. Some of them may be integers, others floats, and yet others may be lists.
  - the elements in a list are always numbered starting from zero.
  - Each of the list's elements may be accessed separately. For example, it can be printed:

```
print(numbers[0]) # accessing the list's first element
```

The len() function The list is a very dynamic entity.

The function takes the list's name as an argument, and returns the number of elements currently stored inside the list (in other words - the list's length).

Removing elements from a list

```
del numbers[1]
```

Note : You can't access an element which doesn't exist

Task (2 min)

```
numbers = [10, 5, 7, 2, 1]

# printing original list content
print("Original list content:", numbers)

# printing previous list content
numbers[0] = 111
print("\nPrevious list content:", numbers)

# copying value of the fifth element to the second
numbers[1] = numbers[4]

# printing current list content
print("New list content:", numbers)

# printing the list's length
print("\nList length:", len(numbers))
```

Introduction to Python

- More about lists

Negative indices are legal

```
numbers = [111, 7, 2, 1]
print(numbers[-1])  # output =1
print(numbers[-2])  #output = 2
```

```
Task ( 3 min)
hatList = [1, 2, 3, 4, 5]  # This is an existing list of numbers hidden in the hat.

# Step 1: write a line of code that prompts the user
# to replace the middle number with an integer number entered by the user.

# Step 2: write a line of code here that removes the last element from the list.

# Step 3: write a line of code here that prints the length of the existing list.

print(hatList)
```

- Functions vs. methods

result = function(arg) result = data.method(arg)

Note: the name of the method is preceded by the name of the data which owns the method. Next, you add a dot, followed by the method name, and a pair of parenthesis enclosing the arguments.

Introduction to Python

- More about lists
  - A new element may be added to the end of the existing list: **list.append(value)**
  - The insert() method can add a new element at any place in the list. **list.insert(location, value)**

follow this code and write the output

```
myList = [] # creating an empty list

for i in range(5):
    myList.append(i + 1)

print(myList)
```

Follow this code and write the output

```
myList = [] # creating an empty list

for i in range(5):
    myList.insert(0, i + 1)

print(myList)
```

```
myList = [10, 1, 8, 3, 5]
total = 0

for i in range(len(myList)):
    total += myList[i]

print(total)
```

```
myList = [10, 1, 8, 3, 5]
total = 0

for i in myList:
    total += i

print(total)
```

Swap

```
variable1 = 1
variable2 = 2
auxiliary = variable1
variable1 = variable2
variable2 = auxiliary
```

Swap in Python

```
var1 = 1
var2 = 2
var1, vari2 = var2, var1
```

Introduction to Python

LAB 2.3

Your task is to:

step 1: create an empty list named beatles;
step 2: use the append() method to add the following members of the band to the list: John Lennon, Paul McCartney, and George Harrison;
step 3: use the for loop and the append() method to prompt the user to add the following members of the band to the list: Stu Sutcliffe, and Pete Best;
step 4: use the del instruction to remove Stu Sutcliffe and Pete Best from the list;
step 5: use the insert() method to add Ringo Starr to the beginning of the list.

Check this code (continue from the last slide)

```
myList = [10, 1, 8, 3, 5]
length = len(myList)
print(length//2)

for i in range(length // 2):
    myList[i], myList[length - i - 1] = myList[length - i - 1], myList[i]

print(myList)
```

```
# step 1
print("Step 1:", beatles)
# step 2
print("Step 2:", beatles)
# step 3
print("Step 3:", beatles)
# step 4
print("Step 4:", beatles)
# step 5
print("Step 5:", beatles)
# testing list length
print("The Fab", len(beatles))
```

Introduction to Python

```
lst = [1, 2, 3, 4, 5]
lst.insert(1, 6)
del lst[0]
lst.append(1)
print(lst)
```

```
lst = []
del lst
print(lst)
```

```
a = 3
b = 1
c = 2
lst = [a, c, b]
lst.sort()
print(lst)
```

```
lst = [1, 2, 3, 4, 5]
lst2 = []
add = 0

for number in lst:
    add += number

lst2.append(add)

print(lst2)
```

```
lst = [1, [2, 3], 4]
print(lst[1])
print(len(lst))
```

```
myList = [8, 10, 6, 2, 4]
myList.sort()
print(myList)
```

```
a = "A"
b = "B"
c = "C"
d = " "

lst = [a, b, c, d]
lst.reverse()

print(lst)
```

Introduction to Python

The inner life of lists

```
list1 = [1]
list2 = list1  # copy list name
list1[0] = 2
print(list2)
output: [2], not [1]
```

slices make a brand new copy of a list, or parts of a list.

```
list1 = [1]
list2 = list1[:] # copy list contents
list1[0] = 2
print(list2)
Its output is [1].
```

One of the most general forms of the slice looks as follows:

myList[start:end]

A slice of this form makes a new (target) list, taking elements from the source list - the elements of the indices from start to end - 1.

```
myList = [10, 8, 6, 4, 2]
newList = myList[1:3]
print(newList)
# OUTPUT [8,6]
```

negative indices

```
myList = [10, 8, 6, 4, 2]
newList = myList[1:-1]
print(newList)
#OUTPUT [ 8, 6, 4]
```

```
myList = [10, 8, 6, 4, 2]
newList = myList[-1:1]
print(newList)
#OUTPUT   []
```

Introduction to Python

1. If you omit the start in your slice, it is assumed that you want to get a slice beginning at the element with index 0.

myList[:end] =myList[0:end]

(1)
```
myList = [10, 8, 6, 4, 2]
newList = myList[:3]
print(newList)
```

2.Similarly, if you omit the end in your slice, it is assumed that you want the slice to end at the element with the index len(myList).

myList[start:] = myList[start:len(myList)]

(2)
```
myList = [10, 8, 6, 4, 2]
newList = myList[3:]
print(newList)
```

3. del instruction is able to delete more than just a list's element at once - it can delete slices too:

(3)
```
myList = [10, 8, 6, 4, 2]
del myList[1:3]
print(myList)
del myList[:]
```

4. The in and not in operators

(4)
```
myList = [0, 3, 12, 8, 2]

print(5 in myList)   # False
print(5 not in myList) # True
print(12 in myList) # True
```

Introduction to Python

- How to use list

find the greater value in the list

```
myList = [17, 3, 11, 5, 1, 9, 7, 15, 13]
largest = myList[0]

for i in range(1, len(myList)):
    if myList[i] > largest:
        largest = myList[i]

print(largest)
```

```
myList = [17, 3, 11, 5, 1, 9, 7, 15, 13]
largest = myList[0]

for i in myList[1:]:
    if i > largest:
        largest = i

print(largest)
```

find the location of a given element inside a list

```
myList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
toFind = 5
found = False

for i in range(len(myList)):
    found = myList[i] == toFind
    if found:
        break

if found:
    print("Element found at index", i)
else:
    print("absent")
```

lottery game

```
drawn = [5, 11, 9, 42, 3, 49]
bets = [3, 7, 11, 42, 34, 49]
hits = 0

for number in bets:
    if number in drawn:
        hits += 1

print(hits)
```

What is the output of the following snippet?

l1 = ["A", "B", "C"] l2 = l1 l3 = l2 del l1[0] del l2[0] print(l3)	l1 = ["A", "B", "C"] l2 = l1[:] l3 = l2[:] del l1[0] del l2[0] print(l3)
l1 = ["A", "B", "C"] l2 = l1 l3 = l2 del l1[0] del l2 print(l3)	Insert in or not in instead of ??? so that the code outputs the expected result. myList = [1, 2, "in", True, "ABC"] print(1 ??? myList) # outputs True print("A" ??? myList) # outputs True print(3 ??? myList) # outputs True print(False ??? myList) # outputs False
l1 = ["A", "B", "C"] l2 = l1 l3 = l2 del l1[0] del l2[:] print(l3)	Insert in or not in instead of ??? so that the code outputs the expected result. ☺ myList = [1, 2, "in", True, "ABC"] print(1 in myList) # outputs True print("A" not in myList) # outputs True print(3 not in myList) # outputs True print(False in myList) # outputs False

Introduction to Python

- LAB 2.4
- Scenario
- Imagine a list - not very long, not very complicated, just a simple list containing some integer numbers. Some of these numbers may be repeated, and this is the clue. We don't want any repetitions. We want them to be removed.
- Your task is to write a program which removes all the number repetitions from the list. The goal is to have a list in which all the numbers appear not more than once.
- Note: assume that the source list is hard-coded inside the code - you don't have to enter it from the keyboard. Of course, you can improve the code and add a part that can carry out a conversation with the user and obtain all the data from her/him.
- Hint: we encourage you to create a new list as a temporary work area - you don't need to update the list in situ.
- We've provided no test data, as that would be too easy. You can use our skeleton instead.

```
myList = [1, 2, 4, 4, 1, 4, 2, 6, 2, 9]
#
# put your code here
#
print("The list with unique elements only:")
print(myList)
```

Introduction to Python

Lists in lists (**list comprehension)**

What is the output of the code?

```
row = []

for i in range(8):
    row.append(WHITE_PAWN)
```

created on-the-fly during program execution, and is not described statically.

```
row = [WHITE_PAWN for i in range(8)]
```

Examples:

```
print ([x ** 2 for x in range(10)])
#OT [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
print ([2 ** i for i in range(8)])
#OT [1, 2, 4, 8, 16, 32, 64, 128]
```

```
squares = [1, 3 , 6, 9]
print ([x for x in squares if x % 2 != 0 ])
#OT [1, 3, 9]
```

Lists in lists: two-dimensional arrays

```
print ([['EMPTY' for i in range(8)] for j in range(8)])
```

```
EMPTY = "-"
ROOK = "ROOK"
board = []

for i in range(8):
    row = [EMPTY for i in range(8)]
    board.append(row)

board[0][0] = ROOK
board[0][7] = ROOK
board[7][0] = ROOK
board[7][7] = ROOK

print(board)
```

Congratulations !

You can deal now with :

- *Boolean values to compare different values and control the execution paths using the if and if-else instructions;*
- *the utilization of loops (while and for) and how to control their behavior using the break and continue instructions;*
- *the difference between logical and bitwise operations;*
- *the concept of lists and list processing, including the iteration provided by the for loop, and slicing;*
- *the idea of multi-dimensional arrays.*