Basic Programming

Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While

Alexandra Ciobotaru

12 nov 2021

Syllabus

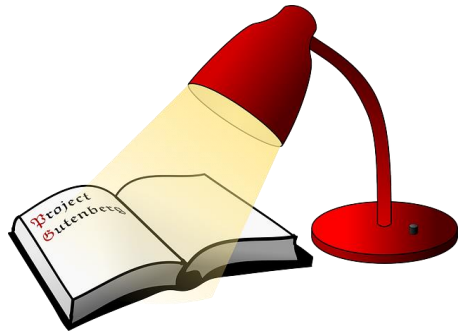
- **Lesson 1. Computers, Programming and Cognitive Science. From pseudocode to programming languages.**
- **Lesson 2. Variables in Python. Basic calculus. Using Math library, type() and help() functions.**
- **Lesson 3. Collections: Lists, Tuples.**
- **Lesson 4. Strings. Working with strings.**
- **Lesson 5. Branching and decisions: Logical operators, If-Statements, Nested conditions. Loops: For and While**
- Lesson 6. Working with matrices in Python. Python numpy library. Quiz 1 (25%).
- Lesson 7. Collections: Dictionaries. JSON construction.
- Lesson 8. Working with files. Reading and Writing.
- Lesson 9. Analyzing dataframes. Pandas and Matplotlib Python libraries.
- Lesson 10. Creating functions. Recursive functions. Quiz 2 (25%).
- Lesson 11. Object-Oriented Programming: Encapsulation, Inheritance and Polymorphism.
- Lesson 12. Object-Oriented Programming (cont.). Error handling. Best practices when programming.
- Lab (20%) + final exam (30%).

What you'll learn today:

- 1. Loops
  - 1.1. The FOR loop
  - 1.2. The WHILE loop
- 2. Conditionals
  - 2.1. If-else statement
  - 2.2. Logical operators
  - 2.3. More decision branches
  - 2.4. Conditionals inside loops
  - 2.5. Break and continue hacks
  - 2.6. Pass statement

1. Loops

- **Looping** means running the same command until a condition is met
- A loop in programming is a set of instructions which you repeat – you repeat them based on a *counter* or a *condition*:
  - Counter loop (**for loop**): you brush your teeth and you go up and down with your brush 10 times
  - Conditional loop (**while loop**): you read a book as long as the condition of having the light on is true (remember when mom used to come to your room and turn off the light?)



https://www.caseydentists.com.au/too-much-toothbrush-time-overbrushing-damages-your-teeth/

1.1. FOR loop

- A for loop executes commands once for each value in a collection

loop variable

collection

```
for i in [0,1,2,3]:
    print("iteration for i = ", i)
```

body

Result:

```
iteration for i =  0
iteration for i =  1
iteration for i =  2
iteration for i =  3
```

- Iterating can be done over strings as well (try it out):

```
a = 'word'
for letter in a:
    print(letter)
```

Range

- The collection used to iterate over can be written as a range of numbers:

```
for i in range(3):
    print(i)
```

Result:

0
1
2

Observe that the last number in range is not included in the enumeration.

```
my_numbers = range(1,11)
for counter in my_numbers:
    print(counter)
```

Result:

1
2
3
4
5
6
7
8
9
10

The accumulator

- A common pattern in programs is to:
  - initialize an *accumulator* variable to zero (an empty list) and
  - update the variable with values from a collection
- Count the sum of the elements in range of a number

```
sum = 0
for elem in range(3):
    sum = sum + elem
print(sum)
```

Result:
3

Explanation:
elem = 0 -> sum = 0 + 0 = 0
elem = 1 -> sum = 0 + 1 = 1
elem = 2 -> sum = 1 + 2 = 3

- Append to a list computed values

```
values = [1, 2, 3, 4]
squared_values = []
for x in values:
    squared_values.append(x**2)
print(squared_values)
```

Result:
[1, 4, 9, 16]

Explanation:
X = 1 -> squared_values = [1]
X = 2 -> squared_values = [1, 4]
X = 3 -> squared_values = [1, 4, 9]
X = 4 -> squared_values = [1, 4, 9, 16]

Exercises

- https://holypython.com/intermediate-python-exercises/exercise-8-python-for-loop/

1.2. WHILE loop

- With the while loop we can execute a set of statements as long as a condition is true.
- Example: print i as long as i is less than 6:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

loop variable

condition

body

i += 1 means i = i+1

Result:

1
2
3
4
5

Explanation:

i = 1
i < 6? Yes -> “1”; i = 1+1 = 2
i < 6? Yes -> “2”; i = 2+1 = 3
i < 6? Yes -> “3”; i = 3+1 = 4
i < 6? Yes -> “4”; i = 4+1 = 5
i < 6? Yes -> “5”; i = 5+1 = 6
i < 6? No

Beware that not upgrading the loop variable will make it loop forever!!

```
i = 1
while i < 6:
  print(i)
  # i += 1
```

2. Conditionals

2.1. IF - ELSE statement

- Remember Booleans variables -> True/False
- The IF statement controls whether a block of code is executed or not:

```
num = 105
if num > 100:
    print(num)
```

condition

Explanation:
is num > 100?
yes -> print it
no -> do nothing

- We use ELSE to execute a block of code when the condition is not met:

```
num = int(input("Insert number: "))
if num > 100:
    print(num, " number is high")
else:
    print(num, " number is not that high")
```

2.2. Logical Operators

- Logical operators are used on conditional statements (either True or False).
- They perform **Logical AND**, **Logical OR** and **Logical NOT** operations:

OPERATOR	DESCRIPTION	SYNTAX
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if operand is false	not x

- Logical conditions - Python supports the usual logical conditions from maths:
  - Equals: a == b
  - Not Equals: a != b
  - Less than: a < b
  - Less than or equal to: a <= b
  - Greater than: a > b
  - Greater than or equal to: a >= b
- These conditions are most commonly used in “if statements”

Examples

- Logical operators

```
a = 200
b = 33
if b > a:
  print("b is greater than a")
else:
  print("b is not greater than a")
```

- And

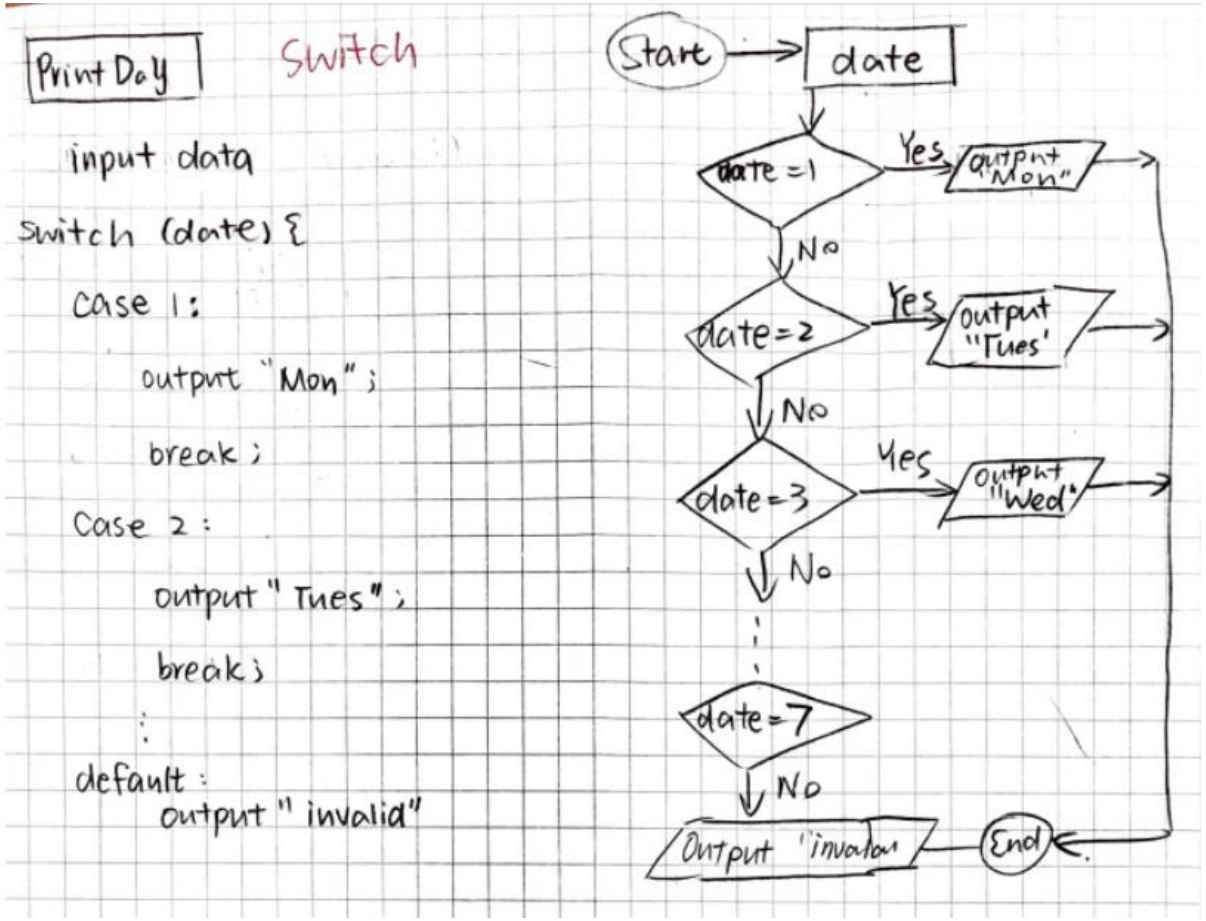
```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
```

- Or

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
```

2.3. If you have more decision branches -> use **elif**

- Remember this pseudocode from Lesson 1?




```
date = int(input("insert date: "))
if date == 1:
    print("Monday")
elif date == 2:
    print("Tuesday")
elif date == 3:
    print("Wednesday")
elif date == 4:
    print("Thursday")
elif date == 5:
    print("Friday")
elif date == 6:
    print("Saturday")
elif date == 7:
    print("Sunday")
else:
    print("invalid number")
```

But you can use **if** as well

```
# difference between if and elif
# A series of ifs execute all lines possible
x = 1
if x==1:
    print("yes")
if x==2:
    print("no")
if x==1 or x==3:
    print("let's see")
else:
    print("wrong")
```

Result:

yes

let's see

```
# if/elifs execute only the first branch with a valid condition
x = 1
if x==1:
    print("yes")
elif x==2:
    print("no")
elif x==1 or x==3:
    print("let's see")
else:
    print("wrong")
```

Result:

yes

2.4. Conditionals inside loops

- Any logical combination is possible

```
num = [20, 43, 12, 88, 97, 110]
for number in num:
    if number > 100:
        print(number, 'is higher than 100')
    else:
        print(number, 'is not that high')
```

Result:

20 is not that high
43 is not that high
12 is not that high
88 is not that high
97 is not that high
110 is higher than 100

- Nested ifs:

```
num = [20, 43, 12, 88, 97, 110, 1000]
for number in num:
    if number > 100:
        print(number, 'is higher than 100')
        if number == 1000:
            print('this', number, 'is really high!')
    else:
        print(number, 'is not that high')
```

Result:

20 is not that high
43 is not that high
12 is not that high
88 is not that high
97 is not that high
110 is higher than 100
1000 is higher than 100
this 1000 is really high!

2.5. Break and continue hacks

- Break is used to exit the loop when a condition is met:

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

Explanation:
i=1; i<6? Yes -> “1”; is i=3? No; i gets the value of 2
i=2; i<6? Yes -> “2”; is i=3? No; i gets the value of 3
i=3; i<6? Yes -> “3”; is i=3? Yes -> exit loop

- With the continue statement we can stop the current iteration, and continue with the next:

```
i = 1
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

Explanation:
i=1; i<6? Yes -> i gets the value of 2; is i=3? No; “2”
i=2; i<6? Yes -> i gets the value of 3; is i=3? Yes; going further (no printing)
i=3; i<6? Yes -> i gets the value of 4; is i=3? No; “4”
i=4; i<6? Yes -> i gets the value of 5; is i=3? No; “5”
i=5; i<6? Yes -> i gets the value of 6; is i=3? No; “6”
I = 6; i<6? No -> exit loop

2.6. Pass statement

- if statements cannot be empty, but if you for some reason have an if statement with no content, put in the pass statement to avoid getting an error
  - `a = 33`
  - `b = 200`

  - `if b > a:`
  - `  pass`

If b > a do nothing

Thank you!