Informatik 1 - Biomedical Engineering

Tutor Session 2 - Branching

Overview

- The if-Statment
- For Loop
- While Loop
- Examples

The if-Statement

If expression_1 is true, then instruction 1 should happen. If expression_2 is true, instruction 2 should happen. If both are not true, the else instruction will be executed.
- Examples:

if expression_1:

```
#instruction 1
```

elif expression_2: #optional

```
#instruction 2
```

else: #optional

```
#else instruction
```

In [ ]:
```
weather = input("How is the weather today (rainy/sunny): ") #user input

if weather == "rainy":   #first if expression
    print("clean your room!")
elif weather == "sunny": #first if else expression
    print("you can go swimming :-)")
else:                    #final else expression
    print("pff. don\’t have a recommendation.")
print("anyhow, watch a movie at night.")
```

- Differnece between "is" and "==":
"==" checks wether the values are the same. "is" is a check for object identity.

In [ ]:
```
a = 10.5
b = 10.5
```

In [ ]:
```
a == b
```

In [ ]:
```
a is b
```

So: use == if you mean the objects should represent the same thing (most common usage) and is if you mean the objects should be in identical pieces of memory.

- Logical operators: "and", "or", "in", "not"

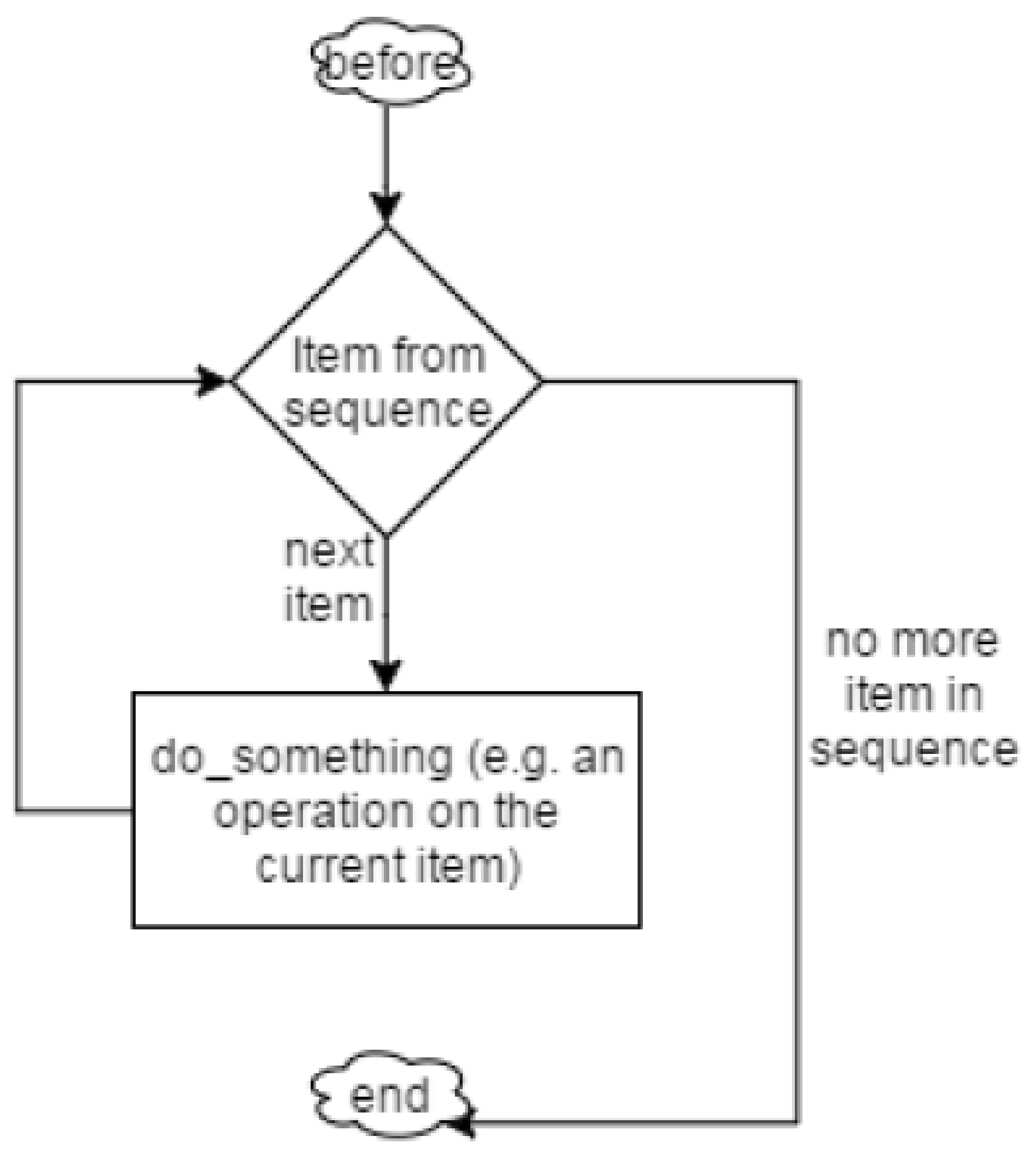
In [ ]:
```
if (True and False) or (not 0): #logical statement
    print ("what???")

#normal logic:                 #explanation
print(True and False)
print(not 0)
print(False or True)
```

In [ ]:
```
#everything but 0 is true
if 0:
    print(True)
else:
    print(False)
```

In [ ]:
```
if "a" in "aha": #the in operator
    print(True)
```

For Loop



for item in sequence: #general usagde of the for loop statement(s)

Using range to iterate:

```
In [ ]: for i in range(0,9): #will print from zero to eight
            print(i)
```

```
In [ ]: for i in range(9):  #exactly the same as above
            print(i)
```

```
In [ ]: for i in range(5,9): #will print from ?? to ??
            print(i)
```

```
In [1]: for i in range(4, 17, 2):
            print(i)
```

```
4
6
8
10
12
14
16
```

```
In [ ]: for i in range(10, 5, -1):
            print(i)
```

```
In [ ]: #summarize even numbers
        res = 0
        for number in range(11): #iterating threw the numbers
            if number % 2 == 0:  #checking for even numbers
                res += number    #summing the evennumbers up
        print(res)               #printing the solution
```

- "break", "pass" and "continue" Statements

You might want to exit a loop completely when a specific condition is triggered or skip a part of the loop and start the next execution. Therefore we have break, pass and continue:

```
In [2]: for letter in "How does break work":    # break
            if letter == 'b':
                break                          #finishes the if-statement and breaks out of the next for-loop, then continues normaly
            print("Letter:", letter)
        print("Finished!")                     #the program continues here after break
```

```
Letter: H
Letter: o
Letter: w
Letter:  
Letter: d
Letter: o
Letter: e
Letter: s
Letter:  
Finished!
```

In [3]:
```
countdown = 10                       # continue
while countdown > 0:
    countdown = countdown -1
    if countdown == 5:
        continue                     # jumps straight to the next loop iteration
    print("countdown:", countdown)   # this is skipped when continue is prompted
print("countdown finished, is something missing?")
```

```
countdown: 9
countdown: 8
countdown: 7
countdown: 6
countdown: 4
countdown: 3
countdown: 2
countdown: 1
countdown: 0
countdown finished, is something missing?
```

In [4]:
```
for num in range(10,20):    # to iterate between 10 to 20
    for i in range(2,num):  # to iterate on the factors of the number
        if num%i == 0:      # to determine the first factor
            j=num/i         # to calculate the second factor
            print("%d equals %d * %d" % (num,i,j))
            break           # to move to the next number, the #first FOR
    else:                   # else part of the loop
        pass                # useless
        print(num,"is a prime number")
```

```
10 equals 2 * 5
11 is a prime number
12 equals 2 * 6
13 is a prime number
14 equals 2 * 7
15 equals 3 * 5
16 equals 2 * 8
17 is a prime number
18 equals 2 * 9
19 is a prime number
```

In [ ]:
```
for letter in "How does pass work":
    if letter == 'p':
        pass  # this is like a spaceholder, if you haven't decided what should happen here...
    print("Letter:", letter)
print("Finished!")
```

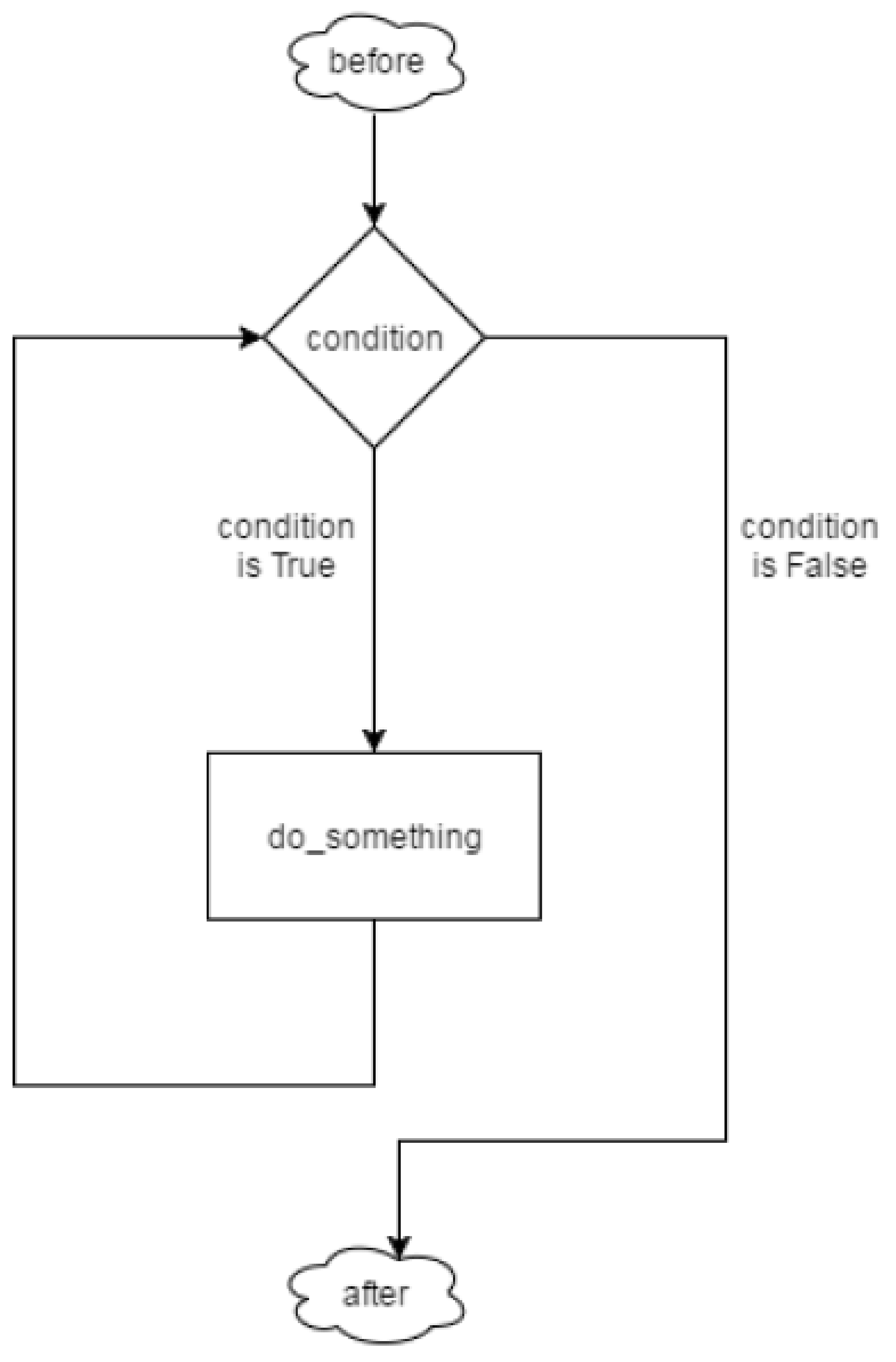
- Iterating over a list:

In [ ]:
```
iteration_list = [1,2,3,4,5]          # a list is created
for i in range(len(iteration_list)):  # Iterating over the length of a list
    print(iteration_list[i])
```

In [ ]:
```
iteration_list = [10, 100, 1000, 10000]  # a list is created
for i in iteration_list:                 # Iterating direktly over the list
    print(i)
```

In [ ]:
```
print(i)
```

While Loop



while condition:

```
# do_something
```

In [ ]:
```python
counter = 3
while counter:
    print(counter)
    counter -= 1
```

In [ ]:
```python
secure_pwd = "123"
user_inp = ""
attempts = 0

while user_inp != secure_pwd:          # the while loop shall continue until the user enters in the correct password
    user_inp = input("Enter pwd: ")    # entering the password
    attempts += 1                      # counting the attempts
print("Authenticated after ", attempts," attempts")  # final output if entering password was successful
```

- The off by one error

In [ ]:
```python
error = "error"
i = 1                  # the counter starts with the first item, but this is not the first letter
while i <= len(error): # Be careful whilst using <= or <
    print(error[i])
    i += 1
```

- Problem with comparing floats

Floating-point numbers are represented in computer hardware as base 2 (binary) fractions. Most decimal fractions cannot be represented exactly as binary fractions. So the decimal floating-point numbers you enter are only approximated by the binary floating-point numbers actually stored in the machine. It is usually unwise to compare two floating numbers with a ==, <= or >=.

In [ ]:
```python
our_sum = 0.0
while our_sum < 1:                  # summing up the number
    our_sum += 0.1
print(our_sum)
if our_sum == 1.1:                     # comparing floats with == will not work
    print("calculation succeeded")
```

Example Program

Calculating π using Leibniz’s formula.

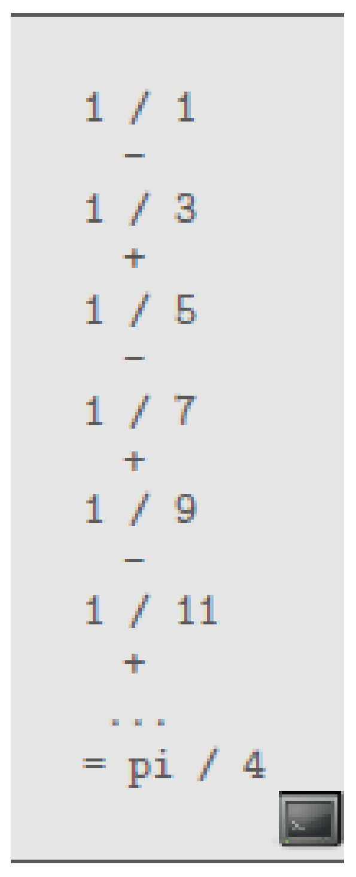
$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}$$

written as a series:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots = \frac{\pi}{4}$$

In [ ]:
```python
#possible solution
ITERATIONS = 1000            # This is a hard coded value in coding standard (ALL LETTERS ARE BIG)
subtotal = 0.0
for n in range(ITERATIONS):  # iterating over the number of iterations defined
    subtotal += (-1)**n / (2*n + 1)  # calculation
pi = subtotal * 4                    # calculation of pi
print("Pi is appr.:", pi)            # printing calculated pi
```

Now try and print out the series: Write a program that prints the first six elements of the Leibniz series to the console. (See picture)



```
1 / 1
  -
1 / 3
  +
1 / 5
  -
1 / 7
  +
1 / 9
  -
1 / 11
  +
 ...
= pi / 4
```

Hint: Use a While Loop (or For Loop) which runs from 1 to 11

In [ ]:
```python
#possible solution:
last_denominator = 11
operator = "-"           # operator starts negative
act = 1
while act <= last_denominator:   # while loop should work until act is bigger than the last_denominator = 11
    print(1, "/", act)    # printing part of the series
    print(" ", operator)  # printing operator
    act += 2              # summing 2 to act

    if operator == "-":   # changing of operator: + and -
        operator = "+"
    else:
        operator = "-"

print(" ...\n= pi / 4")  #printing the final line
```