AA-13-07-2023

1. Git Video ~ TBD
2. Malejące sortowanie:

```
for i in reversed(range(1, 11)):
  print(i)
```

```
fib = [0, 1, 1, 2, 3, 5, 8, 13]
fib.sort(reverse=True)
print(fib)
```

3. Odwrotne sortowanie dla 3 liczb:

```
return result[::-1]
```

4. Lowercase in Python3
   N/A
5.

```
with open("/content/numbers.txt", "r") as file:
```

In Python, the `with` statement is commonly used when working with external resources, such as files, to ensure proper handling and cleanup. The code snippet you provided uses the `with` statement to open a file named "numbers.txt" located in the "/content" directory in read mode.

Let's break down the code line by line:

1. `with open("/content/numbers.txt", "r") as file:`
   - `with open(...)` starts the `with` statement and opens the file specified by the file path "/content/numbers.txt". The `"r"` argument indicates that the file should be opened in read mode.
   - `as file:` assigns the opened file object to the variable named "file". This variable will be used to interact with the file within the `with` block.

Once the file is opened within the `with` block, you can perform various operations on it, such as reading its contents. The advantage of using the `with` statement is that it automatically takes care of closing the file once you're done with it, even if an exception occurs within the block. This ensures that system resources are properly released and prevents any potential issues caused by leaving files open for an extended period.

Powtórzenie wiadomości:

Git

Repo: https://github.com/paciadawid/academy-python

git cheat sheet

- *git config --list* → current setting
- *git init* → creates new repository
- *git clone <REMOTE-URL>* → clones repo from url
- *git add *files** → add changes from local machine to local repository (Git)
- *git commit [*files*] -m "message"* → add changes from local machine to local repository (Git)

git cheat sheet

- *git push -u origin <BRANCH>* → add changes from a local repository (Git) to the remote repository (GitHub)
- *git pull* → pull changes from remote to local repository
- *git fetch -p* → update remote repository branches status
- *git branch <BRANCH-NAME>* → creates new branch
- *git checkout <BRANCH-NAME>* → switch to other branch

Python3

Python basics

a = 1 → int

a = 1.0 → float

a = “1” → string

a = True → boolean

a = [1, “1”] → list

a = (1, “1”) → tuple

a = {1, “1”} → set

a = {“1” : 1} → dict

a = None

Printing

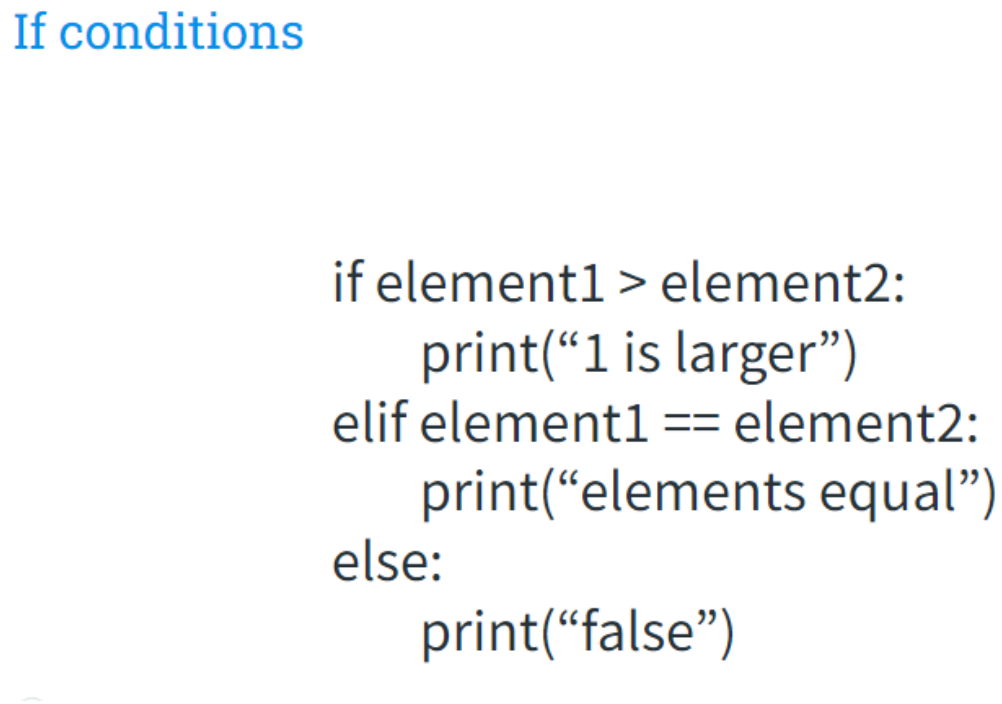
```
hobby = 'swimming'
print("I like " + hobby + " a lot")
print("I like {} a lot".format(hobby))
print(f"I like {hobby} a lot")
```

“My {animal} has {age} years”

```
animal = {"cat": 15, "dog": 7, "duck": 3}


for element in animal:
  print(f"My {element} has {animal[element]}")
```

Wyrażenia warunkowe



Ćw. 1. Przepiszmy kod z IF ~ ELSE na IF ~ ELIF ~ ELSE

```
rok = int(input("Podaj rok: "))



if 0 < rok:
  if (rok % 4 == 0 and rok % 100 != 0) or (rok % 400 == 0):
    print(f"Rok {rok} to rok przestępny")
  else:
    print(f"Rok {rok} to nie jest rok przestępny")
else:
  print(f"Niepoprawny rok")
```

```python
rok = int(input("Podaj rok: "))


if 0 < rok:
  if (rok % 4 == 0 and rok % 100 != 0):
    print(f"Rok {rok} to rok przestępny")
  elif (rok % 400 == 0):
    print(f"Rok {rok} to rok przestępny")
  else:
    print(f"Rok {rok} to nie jest rok przestępny")
else:
  print(f"Niepoprawny rok")
```

Pętle

For loop

```
for i in range(start, stop+1, step):
    print("element")
```

```
for element in list:
    print("element")
```

While loop

```
a, b = 0, 10

while a < b:
    print("a")
    a += 1
```

```
for i in range(10, 20, 2):
    print(i)
print()

a = 10
while a < 20:
    print(a)
    a += 2
```

Ćw. 2. Przeróbmy poniższy kod factorial na sumę liczb od 1 do n

```
n = 4
if n == 0:
    result = 1
else:
    result = 1
    for i in range(1, n + 1):
        result *= i
print(f"Factorial from {n} equals {result}")
```

```python
n = int(input("Podaj dodatnią wartość dla liczby n: "))

sum = 0

if (0 < n):

    sum = n * (n+1) // 2 # O(1)

    print(f"Suma liczb od 1 do {n} wynosi {sum}")

else:

    print("Wartość n powinna być liczbą dodatnią")
```

```python
n = int(input("Podaj dodatnią wartość dla liczby n: "))
sum = 0
if (0 < n):
    for l in range(1, n + 1): #O(n)
        sum += l
    print(f"Suma liczb od 1 do {n} wynosi {sum}")
else:
    print("Wartość n powinna być liczbą dodatnią")
```

http://algorytmy.ency.pl/artykul/notacja_duzego_o

Przykładowe rzędy złożoności

Przykładowe rzędy złożoności funkcji (posortowane rosnąco) to:

- $O(1)$ – złożoność stała,
- $O(\log n)$ – złożoność logarytmiczna,
- $O(n)$ – złożoność liniowa,
- $O(n \log n)$ – złożoność liniowo-logarytmiczna,
- $O(n^2)$ – złożoność kwadratowa,
- $O(n^k)$, gdzie k jest stałą – złożoność wielomianowa,
- $O(k^n)$, gdzie k jest stałą – złożoność wykładnicza,
- $O(n!)$ – złożoność rzędu silnia,

Obsługa wyjątków

Errors handling

```
try:
    code.here()
except (types_of,_exceptions):
    exception.code()
else:
    no.exception()
finally:
    execute.always()
```

```python
try:
    # Code that may raise an exception
    file = open("data.txt", "r")
    # Perform some operations
except FileNotFoundError:
    # Exception handling for file not found
    print("Error: File not found.")
finally:
    # Cleanup code
    file.close()
```

Przetwarzanie napisów

Text parsing

- x.strip()
- x.split(a)
- x.replace(a,b)
- “ ”.join()

- x.upper()
- x.lower()
- x.capitalize()

Ćw. 3. Zaimplementujmy funkcję lowercase() w Python3, która zamieni wszystkie duże litery na małe litery.

ord(A) = 65
A => a : ord(A) + 32
+32 : int
chr(65) => A

```
text = "Ala ma 2 KoTy"
chars = []
count = 0

i = 0
while i < len(text):
    if (ord(text[i]) in range(65, 91)) or (ord(text[i]) in range(97,
123)):
        count += 1
        if text[i] not in chars:
            chars += text[i]
    i += 1
print("Ilość liter:", count)
print("Unikalne litery:", chars)


def lowercase(napis):
    result = ""
    for l in napis:
        if ord(l) in range(65, 91):  # A (CAPITALS)
            result += chr(ord(l)+32) # a (lowers)
        else:
            result += l
    return result

print(text)
print(lowercase(text))
```

```
text = "Ala ma 2 koty"
chars = []
count = 0

i = 0
while i < len(text):
    if (ord(text[i]) in range(65, 91)) or (ord(text[i]) in range(97,
123)):
        count += 1
        if text[i] not in chars:
            chars += text[i]
    i += 1
print("Ilość liter:", count)
print("Unikalne litery:", chars)
```

Operacje na plikach

Files

```
with open("input.txt") as f:
    lines = f.readlines()
```

- f.read()
- f.readline()
- f.readlines()

Funkcje (metody/procedury)

Functions

```
def method(var1, var2):
    logic.here()
    return value
```

```
def sort3(lista: list):
    a = lista[0]
    b = lista[1]
    c = lista[2]

    if (a < b):
        if (a < c):
            if (b < c):
                result = [a, b, c]
            else:
                result = [a, c, b]
        else:
            result = [c, a, b]
    else:
        if (b < c):
            if (a < c):
                result = [b, a, c]
            else:
                result = [b, c, a]
        else:
            result = [c, b, a]


    return result[::-1]

print(sort3([3, 4, 5]))
print(sort3([3, 5, 4]))
print(sort3([4, 3, 5]))
print(sort3([4, 5, 3]))
print(sort3([5, 3, 4]))
print(sort3([5, 4, 3]))
```

```
fib = [0, 1, 1, 2, 3, 5, 8, 13]

print(fib[::-1])
print(fib[3::-1])
print(fib[:4:-1])
print(fib[::-2])


# print(fib[::-1]): This line prints out the reversed version of the fib
list using slicing.
# Slicing is a way to extract a portion of a list (or any iterable) based
on specified indices or a step value.
# In this case, [::-1] is the slicing syntax used to create a reversed
copy of the list. The :: indicates that we want to include all elements,
and the -1 as the step value means we're moving backwards through the
list.
# So, fib[::-1] produces a new list that contains the elements of fib in
reverse order.
```