



ADAM MICKIEWICZ UNIVERSITY IN POZNAŃ

Faculty of English

Robert Dyzman, M.Sc.Eng.

PYTHON PROGRAMMING

CLASS 06



Run „Teams”

Start your IDE

AGENDA:

- Create a file „*class_pp_06.py*”
- Quiz 04
- pass statement + remainder: escape characters, iterables

https://www.w3schools.com/python/ref_keyword_pass.asp

https://www.w3schools.com/python/gloss_python_escape_characters.asp

- DAC algorithm
 - Recursion
 - Exercises
-



EXERCISE 90 (file pp_90.py)

'''

Write a function that will check if a word/sentence is a
palindrome or not.

Use iterative way

These are palindromes, test with them:

Dad

Evil olive.

Never odd or even.

Amore, Roma.

Not palindromes:

test

ad

a

'''



EXERCISE 90 - solution 01

```
def is_palindrome(text):  
    '''  
    input: string  
    output: boolean  
    ,,'  
  
    # your code below  
    text = text.replace(' ', '').replace('.', '  
'').replace(',', '').lower()  
  
    mid = len(text)//2
```



EXERCISE 90-solution 01

```
for i in range(mid):
    if text[i] != text[-i-1]:
        return False
return True

my_text = input('Enter the word or sentence: ...')
if is_palindrome(my_text):
    print(f'{my_text} : is a palindrome!!!')
else:
    print(f'{my_text} : is not a palindrome!!!')
```



EXERCISE 90-solution 02

```
def is_palindrome(text):  
    '''  
    input: string  
    output: boolean  
    ,,'  
    # your code below  
    text = text.replace(' ', '').replace('.', '').replace(',', '  
    ''').lower()  
    reversed_text = text[::-1]  
    if text == reversed_text:  
        return True  
    return False
```



pass statement

In Python, the **pass** statement is used as a placeholder for future code. When the pass statement is executed, nothing happens, but you avoid getting an error when empty code is not allowed.

e.g.,

```
for i in range(1, 5):  
    pass  
  
a, b = 11, 15  
if a < b:  
    pass
```



ESCAPE CHARACTER

**In Python, escape character is used to insert special characters into strings. These special characters are typically illegal within a string, but you can include them using an escape sequence => **

`\'`

`\"`

`\n`

`\t`

`\\`

`s = 'Hey, what's up?' vs s = 'Hey, what\'s up?'`

`s = "His name is \"John\"" vs s = "His name is \\\"John\\\""`

`print(s)`

`print("Multiline strings\\ncan be created\\nusing escape sequences.")`

`print("C:\\Users\\Robert\\Desktop") # regular string`

`print(r"C:\\Users\\Robert\\Desktop") # raw string`



ITERABLES REVISITED

- *iterable* is a container that stores multiple values.
- it can be looped over

e.g.,

range() returns range object which is iterable
string, list, tuples, dictionaries

```
# # 1. range
```

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

```
# #
```

```
# # 2. string
```

```
for x in 'Python':  
    print(x)
```



ITERABLES REVISITED

```
# # 3. list
```

```
for x in ['a', 1, 2, 'b']:  
    print(x)
```

```
# #
```

```
# # 4. tuples
```

```
for x in ('c', 1, 2, 'd'):  
    print(x)
```

```
# #
```

```
# # 5. dictionary
```

```
dic = {'a': 1, 'b': 2, 'c': 3}  
for x in dic:  
    print(x)
```



ITERABLES REVISITED

```
# # over keys()
for x in dic.keys():
    print(x)
# # over values()
for x in dic.values():
    print(x)
# # over items() --> view object
for x in dic.items():
    print(x)
# unpacking iterable
d, c = (5, 6)
print(d, c)
# # over items()
for k, v in dic.items():
    print(k, v)
```



EXERCISE 100

...

Ex. 100

Write a program that computes the value of n factorial - $n!$

Use iterative implementation

Expected results

$$1! = 1$$

$$2! = 1 * 2 = 2$$

$$3! = 1 * 2 * 3 = 6$$

$$10! = 3628800$$

$$32! = 263130836933693530167218012160000000$$

$$n! = 1 * 2 * 3 * \dots * n$$

...



WHAT IS DIVIDE-AND-CONQUER ALGORITHM

- A divide-and-conquer algorithm recursively breaks down a problem into two or more sub-problems of the same or related type, until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem.

(From Wikipedia, the free encyclopedia)



WHAT IS DIVIDE-AND-CONQUER ALGORITHM (DAC)

In DAC we can single out three parts:

1. Divide: This involves dividing the problem into smaller sub-problems.
2. Conquer: Solve sub-problems by calling recursively until solved.
3. Combine: Combine the sub-problems to get the final solution of the whole problem.



WHAT IS RECURSION

- A way to design solutions to problems by **divide-and-conquer algorithm** (DAC)
- A programming technique where a **function calls itself**.
- A recursive definition is made up of two parts.
- There is at least one **base / termination case** that directly specifies the result for a special case.
- There is **at least one recursive case (function calls itself)**, that defines the answer typically in a simpler version of the same problem



MULTIPLICATION ITERATIVE EXAMPLE

- “multiply $a * b$ ” is equivalent to “add a to itself b times”

```
def mult_iter(a, b):
```

```
    result = 0
```

```
    while b > 0:
```

```
        result += a
```

```
        b -= 1
```

```
    return result
```

$$a * b = \underbrace{a + a + a + a + \dots + a}_{b \text{ times}}$$

```
a = int(input('a = '))
```

```
b = int(input('b = '))
```

```
print(f'{a} * {b} = {mult_iter(a,b)}')
```




RECURSIVE WAY OF THINKING

$$a * b = a + a + a + a + a + a + a + a$$

b times

$$a * b = a + a + a + a + a + a + a + a$$

(b - 1) times

$$a * b = a + a * (b - 1)$$

$$a * b = a + \text{mult}(a, b-1)$$

Sub-
problem



MULTIPLICATION RECURSIVE EXAMPLE

- “multiply $a * b$ ” is equivalent to “add a to itself b times”

```
def mult(a, b):
```

```
    if b == 0:
```

```
        return 0
```

```
    if b == 1:
```

```
        return a
```

```
    else:
```

```
        return a + mult(a, b-1)
```

1. What is the base case ? & When it occurs ?

2. How to define recursive case ?

Base
case

Recursive
step

```
a = int(input('a = '))
```

```
b = int(input('b = '))
```

```
print(f'{a} * {b} = {mult(a,b)}')
```



EXERCISE 110 (file pp_110.py)

...

Ex. 100

Write a program that computes the value of n factorial - $n!$

Use recursive implementation

Expected results

$$1! = 1$$

$$2! = 1 * 2 = 2$$

$$3! = 1 * 2 * 3 = 6$$

$$10! = 3628800$$

$$32! = 263130836933693530167218012160000000$$

$$n! = 1 * 2 * 3 * \dots * n$$

...



EXERCISE 120 (file pp_120.py)

'''

Write a function that will check if a word/sentence is a
palindrome or not. (20 min.)

Use recursive implementation

These are palindromes, test with them:

Dad

Evil olive.

Never odd or even.

Amore, Roma.

Not palindromes:

test

ad

a

'''