ADAM MICKIEWICZ UNIVERSITY IN POZNAŃ
Faculty of English

Robert Dyzman, M.Sc.Eng.

PYTHON PROGRAMMING
CLASS 01

Run „Teams”

Start your IDE

Start Moodle, moodle enrolment password : pp@2024

AGENDA:

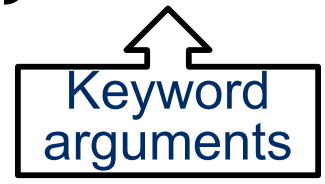
- Final Programming Task analysis
- Create a folder „PYTHON PROGRAMMING”
- Create a file *„class_pp_01.py”*
- Keyword arguments
- Default arguments
- Global / Local variables
- XARGS arguments
- List unpacking
- Copy/Paste to Teams

KEYWORD ARGUMENTS

```
def add_profile(index, ix, iy, area):
    print(
        f'Adding profile {index} with moments
of inertia Ix={ix} cm4, Iy={iy} cm4 and area
{area} cm2’)
```

Keyword arguments are used during function invoke (calling)

```
add_profile('MC014', 166.9, 23.6, 14.99)
add_profile('MC014', ix=166.9, iy=23.6,
area=14.99)
```



DEFAULT (optional) ARGUMENTS

```
print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

How the function will work if you omit the arguments.

„Default arguments” must be declared in function definition.

DEFAULT ARGUMENTS

```
print('1_line', 'Hello', sep='', end=' ')
print('2_line', 'World', sep='')
#
print('1_line', 'Hello')
print('2_line', 'World')
#
def add_item(item_name, quantity=1):
    print(f'{quantity} units of {item_name} was
      added')
add_item('bread')
add_item('apples', 2)
```

Python *args (xargs)

- If you want to create a function with a collection of arguments (and you do not know how many arguments will be passed to your function) add a * before parameter name in function definition;

- You will pass a tuple of arguments

e.g.

Python *args

```
def print_numbers_tuple(*numbers):
    print(numbers)
def print_numbers(*numbers):
    for n in numbers:
        print(n)
def summarize(*numbers):
    sum = 0
    for n in numbers:
        sum += n
    return sum
```

Python *args

```
print_numbers_tuple(1, 2, 3, 4, 5)


print_numbers(1, 2, 3, 4, 5)


print(f'The sum of the numbers is
{summarize(1,2,3,4,5)}')
```

Python **kwargs (xxargs)

- **kwargs in function definition in python is used to pass a keyworded arguments (key and value). Any number argument list;
- We use the name of the argument with the double star;
- A way of creating dictionary (Python will automatically package your arguments into a dictionary);

Python **kwargs (xxargs)

e.g.,

```
def add_user(**user):
    print(user)


def print_user(**user):
    for key, value in user.items():
        print(f'{key} = {value}')
```

Python **kwargs (xxargs)

```
add_user(id=1, Name='John', Surname='Kowalski',
Age=25)

print_user(id=1, Name='John',
Surname='Kowalski', Age=25)
```

*args, **kwargs summary

- https://www.programiz.com/python-programming/args-and-kwargs

Things to Remember:

- `*args` and `**kwargs` are special keyword which allows function to take variable length argument.
- `*args` passes variable number of non-keyworded arguments and on which operation of the tuple can be performed.
- `**kwargs` passes variable number of keyword arguments dictionary to function on which operation of a dictionary can be performed.
- `*args` and `**kwargs` make the function flexible.

VARIABLE SCOPE – general info

- **A variable or function is only available from inside the region it is created. This region is called scope.**
- **Local Scope vs Global Scope**
- **Local Scope : it is a local block of code e.g., inside function, class**
- **Local Variable: a variable created inside a function belongs to the local scope of that function, and can only be used inside that function.**
- **Global Scope : it is a global block of code,**
- **Global Variable: a variable created in the main body of the Python code is a global variable and belongs to the global scope.**
- **Global variables are available from within any scope, global and local.**

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

VARIABLE SCOPE IN FUNCTIONS

```
#1 case, local variable message is not
accessible from outside of the body function

def my_func():
    message = 'a'

print(message)
```

VARIABLE SCOPE

```
#2 case, global variable message is accessible
# in the function body

message = 'a'  # global variable
def my_func():
    print(message)  # works for reading the
                      value, not modifying


my_func()
```

VARIABLE SCOPE

```
# 3 case, variables 'message' are completely
#separate. With this code you can't change
#'message' value from inside the function

message = 'a'  # global variable
def my_func():
    message = 'b’ # now we define a new
                    variable (local)
my_func()
print(message)
```

VARIABLE SCOPE

```
# 4 case, if you want to modify global value from within the
function, use "global" keyword
message = 'a'  # global variable
def my_func():
        '''
        The global keyword ensures that you're working with the
        global variable, not creating a local one with the same
        name
        '''
        global message
        message = 'b'
my_func()
print(message)
```

VARIABLE SCOPE

5 in "if" block of code, variables 'message' are not separate. With this code you can change 'message' value from inside of the if expression

```
message = 'a'  # global variable

if message == 'a':
    message = 'b'

print(message)
```

VARIABLE SCOPE

6 in „loop" block of code, variables 'message' are not separate. With this code you can change 'message' value from inside of the loop expression

```
message = 'a'  # global variable

while message == 'a':
    message = 'b'

print(message)
```

VARIABLE SCOPE - SUMMARY

- Inside a function, can access a variable defined outside;
- Inside a function, cannot modify a variable defined outside unless you use ’global’ keyword;
- Outside a function, cannot access or modify a variable defined inside a function

LIST UNPACKING

```
# unpacking is an operation of assigning
elements of lists to variables
# (this is known to you)
colors = ['red', 'green', 'blue']
var1 = colors[0]
var2 = colors[1]
var3 = colors[2]
print(var1)
```

LIST UNPACKING

- List unpacking is an operation of assigning elements of lists to variables

```
colors = ['red', 'green', 'blue’]
# numbers of the variables on the left side should be equal to the number of the element in the list
var1, var2, var3 = colors
print(var1)
print(var2)
print(var3)
```

LIST UNPACKING

```
# error
numbers = [1, 2, 5, 7, 9, 9, 9, 9, 9, 9]
first_num, second_num = numbers
print(first_num)
print(second_num)
# ok
numbers = [1, 2, 5, 7, 9, 9, 9, 9, 9, 9]
first_num, second_num, *others = numbers
print(first_num)
print(others)
```

Think about *args

LIST UNPACKING

```
#interested in last number
numbers = [1, 2, 5, 7, 9, 9, 9, 9, 9, 9, 100]
first_num, *others, last_num = numbers
print(others)
print(last_num)
```

EXERCISE 10 (file pp_10.py)

```
'''
Write a function named fizz_buzz(num) that will return
the string 'Fizz' if the num parameter  is divisible by
3,
will return the string 'Buzz' if the num is divisible by
'5',
will return the string 'FizzBuzz' if the num is divisible
by 3 and 5
will return num for any other number
Function should include docstring
Program should implement entering the number during
runtime
'''
```