**Filip Konior sprawozdanie nr3**

**Valid or invalid?**

**def print\_two(a, b):  
 print("Arguments: {0} and {1}".format(a, b))  
  
# Valid or invalid?  
#print\_two() invalid  
print\_two(4, 1)  
#print\_two(41) invalid  
#print\_two(a=4, 1) invalid  
#print\_two(4, a=1) invalid  
#print\_two(4, 1, 1) invalid  
#print\_two(b=4, 1) invalid  
print\_two(a=4, b=1)  
print\_two(b=1, a=4)  
#print\_two(1, a=1)  
#print\_two(4, 1, b=1) invalid  
  
print\_two("4","1")  
print\_two([],{})  
#print\_two(,) invalid  
print\_two(4, b=1)**

**Default Arguments**

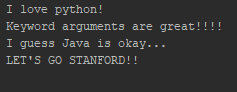
**def keyword\_args(a, b=1, c='X', d=None):  
 print("a:", a)  
 print("b:", b)  
 print("c:", c)  
 print("d:", d)  
  
keyword\_args(5)  
keyword\_args(a=5)  
keyword\_args(5, 8)  
keyword\_args(5, 2, c=4)  
keyword\_args(5, 0, 1)  
keyword\_args(5, 2, d=8, c=4)  
#keyword\_args(5, 2, 0, 1, "") invalid  
#keyword\_args(c=7, 1) invalid  
keyword\_args(c=7, a=1)  
keyword\_args(5, 2, [], 5)  
#keyword\_args(1, 7, e=6) invalid  
keyword\_args(1, c=7)  
#keyword\_args(5, 2, b=4) invalid  
  
#keyword\_args()  
keyword\_args("4")**

**Exploring Variadic Argument lists**

**def variadic(\*args, \*\*kwargs):  
 print("Positional:", args)  
 print("Keyword:", kwargs)  
  
variadic(2, 3, 5, 7)  
variadic(1, 1, n=1)  
#variadic(n=1, 2, 3)  
variadic()  
variadic(cs="Computer Science", pd="Product Design")  
#variadic(cs="Computer Science", cs="CompSci", cs="CS")  
variadic(5, 8, k=1, swap=2)  
variadic(8, \*[3, 4, 5], k=1, \*\*{'a':5, 'b':'x'})  
variadic(\*[8, 3], \*[4, 5], k=1, \*\*{'a':5, 'b':'x'})  
variadic(\*[3, 4, 5], 8, \*(4, 1), k=1, \*\*{'a':5, 'b':'x'})  
variadic({'a':5, 'b':'x'}, \*{'a':5, 'b':'x'}, \*\*{'a':5, 'b':'x'})  
  
variadic(1, 2, \*[3, 4, 5],a=6, \*\*{'n':0, 'm':1})  
variadic(None)**

**speak\_excitedly**

**def speak\_excitedly(msg, i=1, cap=False):  
 if cap:  
 print(msg.upper()+i\*'!')  
 else:  
 print(msg+i\*'!')  
  
speak\_excitedly("I love python")  
speak\_excitedly("Keyword arguments are great",4)  
speak\_excitedly("I guess Java is okay...",i=0)  
speak\_excitedly("Let's go Stanford",2,True)**

****

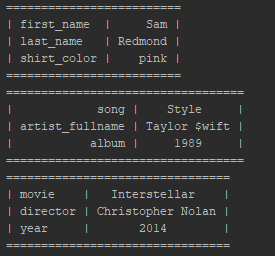
**average**

**def average(\*args):  
 if len(args) == 0:  
 return None  
 else:  
 return sum(args)/len(args)  
  
print(average())  
print(average(5))  
print(average(6, 8, 9, 11))  
  
list = [0,1,2,3,4,5,6,7,8,9,10]  
print(average(\*list))**

****

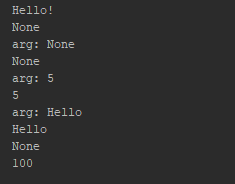
**Challenge: make\_table**

**def make\_table(key\_justify='left',value\_justify='right',\*\*kwargs):  
 #l=max(len(value)+len(key) for key,value in kwargs.items()) # najdluzszy ciag znakow w naszych kwargs  
 l1=max(len(key) for key in kwargs) # najdluzszy element key  
 l2=max(len(value) for key,value in kwargs.items()) # najdluzszy value element  
 print("="\*(l1+l2+7))  
  
 #<, > , ^ setting allinging format  
 def just(x):  
 return {  
 'left':'<',  
 'right':'>',  
 'center':'^'  
 }.get(x,'<') # domyslnie <  
 ky = just(key\_justify)  
 vj = just(value\_justify)  
  
 for key, value in kwargs.items():  
 print("| {:{}{}} | {:{}{}} |".format(key,ky,l1,value,vj,l2))  
 print("=" \* (l1 + l2 + 7))  
  
make\_table(  
 first\_name="Sam",  
 last\_name="Redmond",  
 shirt\_color="pink"  
)  
make\_table(  
 key\_justify="right",  
 value\_justify="center",  
 song="Style",  
 artist\_fullname="Taylor $wift",  
 album="1989"  
)  
make\_table(  
 key\_justify="left",  
 value\_justify="center",  
 movie="Interstellar",  
 director="Christopher Nolan",  
 year="2014"  
)**

****

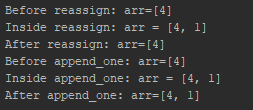
**Function Nuances**

**def say\_hello():  
 print("Hello!")  
  
print(say\_hello())  
  
def echo(arg=None):  
 print("arg:", arg)  
 return arg  
  
print(echo())  
print(echo(5))  
print(echo("Hello"))  
  
def drive(has\_car):  
 if not has\_car:  
 return  
 return 100 # miles  
  
print(drive(False))  
print(drive(True))**

****

**Parameters and Object Reference**

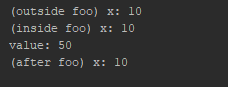
**def reassign(arr):  
 arr = [4, 1]  
 print("Inside reassign: arr = {}".format(arr))  
  
def append\_one(arr):  
 arr.append(1)  
 print("Inside append\_one: arr = {}".format(arr))  
  
l = [4]  
print("Before reassign: arr={}".format(l))  
reassign(l)  
print("After reassign: arr={}".format(l))  
  
l = [4]  
print("Before append\_one: arr={}".format(l))  
append\_one(l)  
print("After append\_one: arr={}".format(l))**

****

**Scope**

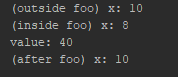
**Case1**

**x = 10  
  
def foo():  
 print("(inside foo) x:", x) #10  
 y = 5  
 print('value:', x \* y) #50  
  
print("(outside foo) x:", x) # 10  
foo()  
print("(after foo) x:", x) #10**

****

**Case2**

**x = 10  
  
def foo():  
 x = 8 # Only added this line - everything else is the same  
 print("(inside foo) x:", x) # 8  
 y = 5  
 print('value:', x \* y) # 40  
  
print("(outside foo) x:", x) # 10  
foo()  
print("(after foo) x:", x) # 10**

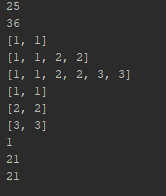
****

**UnboundLocalError**

**x = 10  
  
def foo():  
 print("(inside foo) x:", x) # We swapped this line  
 x = 8 # with this one  
 y = 5  
 print('value:', x \* y)  
  
print("(outside foo) x:", x)  
foo()  
print("(after foo) x:", x)  
  
lst = [1,2,3]  
def foo():  
 lst.append(4)  
foo()**

**Default Mutable Arguments - A Dangerous Game**

**x = 5  
  
def square(num=x):  
 return num \* num  
  
x = 6  
print(square()) # => 25, not 36  
print(square(x)) # => 36  
  
  
def append\_twice(a, lst=[]):  
 lst.append(a)  
 lst.append(a)  
 return lst  
  
  
# Works well when the keyword is provided  
append\_twice(1, lst=[4]) # => [4, 1, 1]  
append\_twice(11, lst=[2, 3, 5, 7]) # => [2, 3, 5, 7, 11, 11]  
  
# But what happens here?  
print(append\_twice(1))  
print(append\_twice(2))  
print(append\_twice(3))  
  
def append\_twice2(a, lst=None):  
 if lst is None:  
 lst = []  
 lst.append(a)  
 lst.append(a)  
 return lst  
print(append\_twice2(1))  
print(append\_twice2(2))  
print(append\_twice2(3))  
  
def fib(n, cache={0: 1, 1: 1}):  
 if n in cache: # Note: default value captures our base cases  
 return cache[n]  
 out = fib(n-1) + fib(n-2)  
 cache[n] = out  
 return out  
  
print(fib(1))  
print(fib(7))  
print(fib(7))**

****

**Investigating Function Objects**

**Default Values (\_\_defaults\_\_ and \_\_kwdefaults\_\_)**

**def all\_together(x, y,z=1, \*nums, indent=True, spaces=4, \*\*options): pass  
  
print(all\_together.\_\_defaults\_\_) # => (1, )  
print(all\_together.\_\_kwdefaults\_\_ ) # => {'indent':True, 'spaces':4}  
  
#Documentation (\_\_doc\_\_)  
def my\_function():  
 *"""Summary line: do nothing, but document it.  
 Description: No, really, it doesn't do anything.  
 """* pass  
  
  
print(my\_function.\_\_doc\_\_)  
# Summary line: Do nothing, but document it.  
#  
# Description: No, really, it doesn't do anything.  
def all\_together(x, y, z=1, \*nums, indent=True, spaces=4, \*\*options):  
 *"""A useless comment"""* print(x + y \* z)  
 print(sum(nums))  
 for k, v in options.items():  
 if indent:  
 print("{}\t{}".format(k, v))  
 else:  
 print("{}{}{}".format(k, " " \* spaces, v))  
  
  
code = all\_together.\_\_code\_\_  
print(code.co\_consts,code.co\_names,code.co\_varnames,code.co\_argcount,sep="\n")**

def nice(): print("You're awesome!")  
def mean(): print("You're... not awesome. OOOOH")

nice.\_\_code\_\_ = mean.\_\_code\_\_  
nice() # prints "You're... not awesome. OOOOH"  
  
#dis module  
def gcd(a, b):  
 while b:  
 a, b = b, a % b  
 return a  
  
  
import dis  
  
dis.dis(gcd)

def annotated(a: int, b: str) -> list:  
 return [a, b]  
  
print(annotated.\_\_annotations\_\_)  
# {'b': <class 'str'>, 'a': <class 'int'>, 'return': <class 'list'>}  
  
#Call (\_\_call\_\_)  
def greet(): print("Hello world!")  
  
greet() # "Hello world!"  
# is just syntactic sugar for  
greet.\_\_call\_\_() # "Hello world!