**Алгоритмы и структуры данных в языке Python.**

**Тест по теме:** **Элементы функционального программирования**

**1.**Имеются функции

def f1(f, a=''):

print(f(a))

def f2(x):

return str(x)

Корректными вызовами функции f1 являются:

1. -: f1(f2, [])
2. -: f1(f2(), [])
3. -: f1(f2([]))
4. -: f1(lambda x: f2(x))
5. -: f1(lambda x: x+1, 5)

**2.** Имеется функция

def f1(a=0, f=str):

print(f(a))

Корректными вызовами функции f1 являются:

1. -: f1()
2. -: f1(f=math.sin())
3. -: f1([2, 3, 5])
4. -: f1([3, 10, 2], max())
5. -: f1(input())

**3.** Имеется программа

def f1(a, b, f):

print(f(a, b))

x=1

y=2

Корректными вызовами функции f1 являются:

1. -: f1(2, 5)
2. -: f1( a=2, b=3, f=sum)
3. -: f1([2, 3, 5], y, list.count)
4. -: f1(3, 10, max)
5. -: f1(x, y, lambda: x+y)

**4.** Имеются функции

def f1(a, f):

print(f(a), f(a))

def f2(x):

return x+1

Корректными вызовами функции f1 являются:

1. -: f1(11, f2(3))
2. -: f1('123', len)
3. -: f1(math.pi, math.cos())
4. -: f1({3, 5}, str)
5. -: f1(5, lambda x: str(x-1))

**5.**Имеется программа

import math

import random

def f1(f):

print(f() + 1)

def f2(x=0):

return x+1

Корректными вызовами функции f1 являются:

1. -: f1()
2. -: f1(f2)
3. -: f1(f2(2))
4. -: f1(random.random)
5. -: f1(math.pi)

**6.** После выполнения операторов

arr = ['5.11.1999', '1.8.2006', '14.6.1980']

arr.sort(key=lambda x: x.split('.')[1])

print(arr)

на экран будет выведено …

1. -: ['14.6.1980', '5.11.1999', '1.8.2006']
2. -: ['1.8.2006', '5.11.1999', '14.6.1980']
3. -: ['14.6.1980', '1.8.2006', '5.11.1999']
4. -: ['5.11.1999', '14.6.1980', '1.8.2006']
5. -: сообщение об ошибке

**7.** После выполнения операторов

arr = [('яблоки', 95, 1), ('молоко', 77, 1), ('чипсы', 85, 2)]

arr.sort(key=lambda x: -x[1]\*x[2])

print(arr)

на экран будет выведено …

1. -: [('молоко', 77, 1), ('чипсы', 85, 2), ('яблоки', 95, 1)]
2. -: [('молоко', 77, 1), ('яблоки', 95, 1), ('чипсы', 85, 2)]
3. -: [('яблоки', 95, 1), ('чипсы', 85, 2), ('молоко', 77, 1)]
4. -: [('чипсы', 85, 2), ('яблоки', 95, 1), ('молоко', 77, 1)]
5. -: сообщение об ошибке

**8.** После выполнения операторов

arr = [(3, 0, 5), (1, 2, 3), (0, 3, 2)]

arr.sort(key=lambda x, y, z: x + y + z)

print(arr)

на экран будет выведено …

1. -: [(3, 0, 5), (1, 2, 3), (0, 3, 2)]
2. -: [(0, 3, 2), (3, 0, 5), (1, 2, 3)]
3. -: [(0, 3, 2), (1, 2, 3), (3, 0, 5)]
4. -: [(1, 2, 3), (3, 0, 5), (0, 3, 2)]
5. -: сообщение об ошибке

**9.** После выполнения оператора

a = sum(map(lambda x: x if x>0 else -x, range(-2, 2)))

значение a равно 4

**10.**После выполнения операторов

ls = [(1, 0), (2, 3), (3, -1), (4, 0), (3, 10)]

f = lambda x: bool(x[1])

a = sum(map(f, filter(f, ls)))

значение a равно 3

**11.** После выполнения операторов

lst = [5, 1, -1, 2]

b = sum(map(lambda x: 2 \* x, filter(lambda x: 0 < x < 3, lst)))

значение b равно 6

**12.** После выполнения операторов

from functools import reduce

lst = [-3, 1, -5, 0]

b = reduce(lambda a, b: a + b if (b > 0) else a, lst )

значение b равно -2

**13.** После выполнения операторов

from functools import reduce

lst = ["++\*", "\*##", "%$#"]

print(reduce(lambda a, b: a + lst[b][b] , range(3), ""))

будет напечатано +##

**14.** После выполнения операторов

ls1 = [0, 2, 0, 1]

ls2 = [7, -2, 0, -1]

b = any(sum(a) for a in zip(ls1, ls2))

значение b равно …

1. -: True
2. -: False
3. -: -0
4. -: 7
5. -: случайное число из 0 и 7

**15.** После выполнения операторов

ls1 = [-5, 7, 10, 0]

ls2 = [2, 0, -2, 1]

b = all(max(a) for a in zip(ls1, ls2))

значение b равно …

1. -: True
2. -: False
3. -: 10
4. -: 2
5. -: [2, 7, 10, 1]

**16.** После выполнения операторов

ls1 = [2, 3]

ls2 = [1, 4]

b = sum((list(a) for a in zip(ls1, ls2)), [])

значение b равно …

1. -: 10
2. -: [3, 7]
3. -: [5, 5]
4. -: [2, 1, 3, 4]
5. -: [[2, 3], [1, 4], []]

**17.** Декоратор - это…

1. -: функция, которая позволяет расширить функциональность другой функции без непосредственного изменения её кода.
2. -: объект, который позволяет поочередно перебирать элементы
3. -: класс, который используется для создания итерируемого объекта
4. -: функция, содержащая выражение yield
5. -: функция, в которой есть вложенная функция

**18.** Итераторами являются функции:

1. -: enumerate()
2. -: zip()
3. -: open()
4. -: max()
5. -: reduce()

**19.** Справедливы следующие утверждения:

1. -: метод \_\_iter\_\_() должен возвращать итератор
2. -: параметром функции next() является итератор
3. -: параметром функции next() является итерируемый объект
4. -: параметром функции iter() является итератор
5. -: параметром функции iter() является итерируемый объект

**20.** Функция-генератор:

1. -: при повторном обращении продолжает своё исполнение с места, на котором была приостановлена
2. -: обязательно содержит инструкцию, которая возбуждает исключение StopIteration
3. -: обязательно содержит хотя бы одну инструкцию yield
4. -: не сохраняет значения переменных между вызовами
5. -: не может содержать инструкцию return

**21.** Для выражений-генераторов справедливы следующие утверждения:

1. -: после прохождения по выражению оно остается пустым
2. -: можно распечатать элементы функцией print()
3. -: можно получить длину функцией len()
4. -: может быть бесконечным
5. -: нельзя писать без скобок

**22.** Что будет выведено на экран в результате выполнения следующей программы:

def addClosure(val1):

def closure(val2):

return val1 + val2

return closure

class AddFunctor(object):

def \_\_init\_\_(self, val1):

self.val1 = val1

def \_\_call\_\_(self, val2):

return self.val1 + val2

cl = addClosure(2)

fn = AddFunctor(2)

print (cl(1), fn(1))

1. -: 3
2. -: 2
3. -: 2 3
4. -: 3 3
5. -: произойдет ошибка

**23.** Что будет выведено на экран в результате выполнения следующей программы в Python 3:

def f(x, y, z, a=None, b=None):

print (x, y, z, a, b)

apply(f, [1, 2, 3], {'a': 4, 'b': 5})

1. -: 1, 2, 3, 4, 5
2. -: 1 2 3 4 5
3. -: 1 2 3 None None
4. -: сообщение об ошибке
5. -: 1 2 3 a b

**24.** Что будет выведено на экран в результате исполнения следующего кода:

from functools import reduce

numbers = [2, 3, 4, 5, 6]

reduce(lambda res, x: res\*x, numbers, 0)

1. -: 720
2. -: 0
3. -: сообщение об ошибке
4. -: [6 12 20 30]
5. -: <reduce…>

**25.** В результате выполнения следующей программы:

from functools import reduce

s = "0###12#34##"

v = reduce(lambda x, y: x+len(y), filter(None, s.split("#")), 0)

v

на экран будет выведено 5

**26.** После выполнения операторов

lst = [[1, 2, 3], [2, 3, 4], [0, 1, 1, 1], [0, 7]]

t = max(map(lambda x: x[-2], filter(lambda x: len(x)>2, lst)))

значение t равно 3

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**27.** После выполнения операторов

li = [7, -16, 2, -11]

x = all(min(y) for y in enumerate(li, 3))

значение x будет равно …

1. -: True
2. -: False
3. -: -16
4. -: -11
5. -: None

**28.** В результате выполнение следующей программы:

itr = iter(sorted('PYTHON'))

next(itr)\*2

print(''.join(list(itr)))

на экран будет выведено

1. -: [ 'O', 'P', 'T', 'Y']
2. -: NOPTY
3. -: HNOPTY
4. -: сообщение об ошибке
5. -: OPTY

**29.** Пусть задана следующая функция:

def decorator\_function(func):

def wrapper():

print('1', end='')

func()

return wrapper

Что будет выведено на экран в результате выполнения следующей программы:

@decorator\_function

def fun():

print('2')

fun()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12

**30.** Имеется генератор

gn = (x\*\*2+x+1 for x in range(5) if x%2==0)

next(gn)

Что будет выведено на экран при выполнении инструкции «print(min (gn))» 1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_