Лабораторная работа № 2: Операторы ветвления, циклы, списки и строки

1. Создайте новую программу **lab_02_01.py** по следующему шаблону:

```
'''
    Условия
'''
# if..else
num = int(input("How many times have you been to the Hermitage? "))
if num > 0:
    print("Wonderful!")
    print("I hope you liked this museum!")
else:
    print("You should definitely visit the Hermitage!")

# if..elif..else
course = int(input("What is your course number?"))
if course == 1:
    print("You are just at the beginning!")
elif course == 2:
    print("You learned many things, but not all of them!")
elif course == 3:
    print("The basic course is over, it's time for professional disciplines!")
else:
    print("Oh! You need to hurry! June is the month of thesis defense")

x = 5
y = 12
if y % x > 0 : print("%d cannot be evenly divided by %d" % (y,x))

z = 3
x = "{} is a divider of {}".format(z,y) if y%z==0 else "{} is not a divider of {}".format(z,y)
print(x)
```

```
print("\n\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Пример результата выполнения программы приведен ниже:

```
How many times have you been to the Hermitage?  2
Wonderful!
I hope you liked this museum!
What is your course number? 1
You are just at the beginning!
12 cannot be evenly divided by 5
3 is a divider of 12
```

Условный оператор

В языке программирования Python существует лишь один тип оператора ветвления: **if** … **elif** … **else**:

```
if <условие>:
    <операторы>
[elif <условие>:
    <операторы>]
[else:
    <операторы>]
```

В квадратных скобках **[ ]** указаны необязательные блоки. В качестве проверяемого условия (**<условие>**) может выступать простое или составное логическое выражение, использующее различные логические операции. Одним из основных отличий языка программирования Python от других языков программирования является отсутствие явных границ для вложенных операторов (**<операторы>**) в условиях, циклах и других операторах и блоках. В Python уровень вложенности показывается с помощью отступов, указанных перед тем или иным оператором. Вложенность блока операций в **if** или **else**, определяется отступом перед каждым из операторов.

В случае, если оператор ветвления содержит лишь один вложенный оператор, он может быть записан в следующем виде:

```
if a>b: c=1
```

Для проверки выполнения условия также может быть использован тернарный оператор ветвления, т.е. сокращенная форма условия, который представляется в следующем формате:

```
c = 1 if a>b else 0
```

Следует отметить, что в языке программирования Python отсутствует возможность использования **switch..case**.

2. Дополните код программы **lab_02_01.py**. Создайте переменную **p**, в которую будет записано значение количества выполненных за год лабораторных по различным дисциплинам. Осуществите вывод значения переменной в терминал, если значение больше 10. Оператор ветвления запишите двумя способами: в одну строку и в несколько строк. Ознакомьтесь с результатом.

3. Дополните код программы **lab_02_01.py**, создав переменные **a** со значением 157 и **b** со значением 525. Осуществите проверку следующих условий и выполнение соответствующих действий:

- если **a>b**, рассчитайте остаток от деления **a** на **b** и выведите значение на экран;
- если **a<b**, рассчитайте остаток от деления **b** на **a** и выведите значение на экран;
- если **a==b**, рассчитайте произведение чисел **a** и **b** и выведите значение на экран.

Осуществите проверку, изменяя значения переменных в соответствии с условиями. Ознакомьтесь с результатом.

4. Создайте новую программу **lab_02_02.py** по следующему шаблону:

```
'''
    Циклы
'''
# while
print("Numbers < 10 (while):")
i = 0
while (i<10):
    print(i, end=" ") # print in one line
    i += 1
print("\n")
```

```
# for
print("Numbers < 10 (for):")
for i in range(0,10):
    print(i, end=" ")
else:
    print("\nThe next number is 10\n")

# break
sum = 0
for i in range(0,100):
    if i > 10:
        print("\nWe reached the end, final sum: ",
sum)
        break
    sum += i

# continue
i = 0
while i<=15:
    if i % 3 == 0:
        i += 1
        continue
    print(i, end=" ")
    i += 1

print("\n")

# pass
print("Let's print numbers again!")
for i in range(0,10):
    pass
    print(i, end=" ")

print("\n\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
Numbers < 10 (while):
0 1 2 3 4 5 6 7 8 9
```

```
Numbers < 10 (for):
0 1 2 3 4 5 6 7 8 9
The next number is 10

We reached the end, final sum:  55
1 2 4 5 7 8 10 11 13 14

Let's print numbers again!
0 1 2 3 4 5 6 7 8 9
```

Циклы

В языке программирования Python существует два типа циклов: цикл с итератором и цикл с предусловием. Вложенные операторы должны быть обозначены отступом. Цикл с итератором представляется в следующем виде:

```
for <переменная> in <объект>:
    <операторы>
```

Здесь **<переменная>** – переменная-итератор, которая представляет собой определенный объект или элемент из списка (**<объект>**), одно из свойств объекта (**<объект>**) и т.д. По окончании выполнения вложенных операторов (**<операторы>**) и переходе к следующей итерации, происходит переход к следующему объекту, свойству и т.д. Например, для написания цикла на 10 итераций можно использовать запись: **for i in range(0,10):** …, где **range** – функция получения списка значений заданного интервала.

Цикл с предусловием реализуется в следующем формате:

```
while <условие>:
    <операторы>
```

Одним из основных отличий языка Python является возможность использования **else** для циклов обоих видов, что позволяет определить операторы, которые будут выполнены, если условие цикла дало ложный результат. Для цикла с предусловием данный блок указывается следующим образом:

```
while <условие>:
    <операторы>
else:
    <операторы>
```

Также внутри циклов могут быть использованы следующие ключевые слова:

- **break** – остановка цикла
- **continue** – переход на следующую итерацию цикла
- **pass** – не несет смыслового значения, может быть использован для упрощения идентификации части кода, которую необходимо дополнить позже

5. Дополните код программы **lab_02_02.py**. Осуществите вывод на экран чисел в диапазоне от 0 до 500, кратных 7, с использованием двух видов циклов. По окончании работы циклов в блоке **else** выведите сообщение **"All numbers were printed!"**. Ознакомьтесь с результатом.

6. Дополните код программы **lab_02_02.py**. Модифицируйте написанные на предыдущем шаге циклы, добавив пропуск вывода значений кратных 14 и остановку цикла по достижении значения 300 (с помощью **break**). Ознакомьтесь с результатом.

7. Дополните код программы **lab_02_02.py**. Осуществите вывод на экран следующей таблицы с использованием двух видов циклов:

```
1 0 0 0
0 2 0 0
0 0 3 0
0 0 0 4
```

8. Создайте новую программу **lab_02_03.py** по следующему шаблону:

```
'''
    Списки
'''
a = [1,2,3,4,5]
print("a[0]: ", a[0])
print("List a[0:5]: ", a[0:5])
```

```
print("List a[:]: ", a[:])
print("List a: ", a)
print("List a[1:5]: ", a[1:])
print("List a[0:4]: ", a[:4])
print("Length of list a: ", len(a))

print("\nList a (by index):")
# получение элементов списка в цикле (1)
for i in range(0,len(a)):
    print(a[i], end=" ")
print("\nList a (by elements):")
# получение элементов списка в цикле (2)
for elem in a:
    print(elem, end=" ")
print("\n")

b = []
b.append(20) # добавление элемента в конец
b.extend(a) # добавление элементов списка a в b
print("List b (extended): ",b)
b.insert(3,5) # добавить элемент на позицию
print("List b (insert element): ",b)
b.remove(5) # удалить первый элемент, равный 5
print("List b (remove element): ",b)
c1 = b.pop() # удалить и вернуть последний элемент
print("List b (pop last element): ",b)
print("c1: ",c1)
c2 = b.pop(3) # удалить и вернуть эл. с позиции
print("List b (pop 3rd element): ",b)
print("c2: ",c2)
bCopy = b.copy() # создать копию списка
print("List b: ",b)
print("List bCopy: ",bCopy)
b.reverse() # поменять элем. местами с конца в начало
print("List b (reversed): ",b)
b.sort(reverse=True) # сортировка элементов списка
print("List b (sorted): ",b)
b.clear(); # очистка списка
print("List b (cleared): ",b)
print("\n")

# сравнение списков
a1 = [1,2,4]
a2 = [1,2,3]
```

```
print("a1 == a2: ", a1 == a2)
a1.append(-1)
print("a1 == a2: ", a1 == a2)
print("a1 > a2: ", a1 > a2)
print("a1 < a2: ", a1 < a2)
print("\n\n")
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
a[0]:  1
List a[0:5]:  [1, 2, 3, 4, 5]
List a[:]:  [1, 2, 3, 4, 5]
List a:  [1, 2, 3, 4, 5]
List a[1:5]:  [2, 3, 4, 5]
List a[0:4]:  [1, 2, 3, 4]
Length of list a:  5

List a (by index):
1 2 3 4 5
List a (by elements):
1 2 3 4 5

List b (extended):  [20, 1, 2, 3, 4, 5]
List b (insert element):  [20, 1, 2, 5, 3, 4, 5]
List b (remove element):  [20, 1, 2, 3, 4, 5]
List b (pop last element):  [20, 1, 2, 3, 4]
c1:  5
List b (pop 3rd element):  [20, 1, 2, 4]
c2:  3
List b:  [20, 1, 2, 4]
List bCopy:  [20, 1, 2, 4]
List b (reversed):  [4, 2, 1, 20]
List b (sorted):  [20, 4, 2, 1]
List b (cleared):  []

a1 == a2:  False
a1 == a2:  False
a1 > a2:  True
a1 < a2:  False
```

Списки

Одним из основных типов данных в языке программирования Python являются списки (Lists) – индексированные совокупности переменных разного типа. Новый пустой список может быть создан с использованием любого из следующих операторов:

```
a = []
a = list()
```

Для создания заполненного списка, его значения указываются в квадратных скобках:

```
a = [1, 2, 3, "keyboard", "mouse", [0,1] ]
```

Получение количества элементов списка осуществляется с помощью функции **`len(a)`**, где **`a`** – имя списка. Доступ к элементам списка осуществляется по индексу. Могут быть использованы следующие форматы записи, в зависимости от поставленной задачи:

- **`a[0]` – получение элемента на позиции 0**
- **`a[0:5]` – получение первых пяти элементов**
- **`a[-1]` – получение последнего элемента списка**
- **`a[-2]`** – получение элемента, находящегося на 2-й позиции с конца списка
- **`a[:3]`** – получение элементов от начала списка до 3 позиции
- **`a[2:]`** – получение элементов от 2 позиции до конца списка
- **`a[:]`** – получение всех элементов списка
- **`a[0:len(a)]`** – получение всех элементов списка
- **`a`** – получение всех элементов списка, например: **`print(a)`**

Для удаления элемента из списка используется ключевое слово **`del`**, например, **`del a[3]`** для удаления элемента списка или **`del a`** для удаления списка целиком.

Для работы со списком могут быть использованы функции, описанные далее в таблице. В квадратных скобках указаны аргументы, которые могут быть опущены.

Функция	Описание	Пример
append(*element*)	Добавление элемента ***element*** в конец списка	**a.append(1)**
extend(*list*)	Добавление в список элементов списка ***list***	**a.extend([2,3,4])**
insert(*index*,*element*)	Вставка элемента ***element*** на позицию ***index***	**a.insert(1,23)**
remove(*element*)	Удаление первого встретившегося элемента ***element***	**a.remove(3)**
pop([*index*])	Удаление и возвращение последнего элемента списка (если функция вызвана без аргументов) или элемента с указанной позиции ***index***	**el = a.pop()** **el = a.pop(2)**
copy()	Создание и возвращение копии элементов списка	**aC = a.copy()**
reverse()	Изменение положения элементов списка (и конца в начало)	**a.reverse()**
sort([*reverse={True|False}*])	Сортировка элементов по возрастанию (***reverse=False***) или по убыванию (***reverse=True***)	**a.sort()**
clear()	Очистка списка: удаление всех элементов из списка	**a.clear()**

В таблице приведены основные функции. Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном структурам данных:
https://docs.python.org/3/tutorial/datastructures.html

Для сравнения списков используются стандартные операторы сравнения. Элементы списка поочередно сравниваются между собой по лексикографическому принципу. Если все элементы списков равны между собой – списки считаются равными.

9. Дополните код программы **lab_02_03.py**. Создайте список **list1**, заполненный 10 значениями, введенными с клавиатуры. Отсортируйте список по убыванию, используя стандартную функцию, после чего выведите последние пять значений списка на экран.

10. Дополните код программы **lab_02_03.py**. Создайте список **list2**, являющий копией списка **list1**, значения которого перенесены из конца списка в начало. Используйте стандартные функции. Выведите список **list2** на экран до и после реверсирования значений.

11. Создайте новую программу **lab_02_04.py** по следующему шаблону:

```
'''
    Строки
'''
group = input("What is your group number? ")
print("Your group is ",group)
print("\nString:")
# получение символов строки в цикле (1)
for i in range(0,len(group)):
    print(group[i], end=" ")
print("\nString:")
# получение символов строки в цикле (2)
for elem in group:
    print(elem, end=" ")
print("\n")

s = "Hello, World!"
print(s)
num = s.count('o')
print("Count 'o': ",num)

ind = s.find("world")
print("Find 'world' (by find()): ",ind)
ind = s.find("World")
print("Find 'World' (by find()): ",ind)
ind = s.index("World")
print("Find 'World' (by index()): ",ind)
print("Find 'World' (by in operation): ", "World" in
s)
print("\nReplace substring:")
```

```
s1 = s.replace("Hello", "hello")
print(" Old: {} \n New: {} ".format(s, s1))
l1 = s.split(",")
print("List l1 (splitted): ",l1)
l2 = s.partition(",")
print("List l2 (partitioned): ",l2)
s2 = s
print("Copy s in s2; s2: ",s2)
sep = "|"
l = ["h","e","l","l","o"]
s = sep.join(l)
print(s)
print("\n")

st = "Привет, мир!"
st1 = st.encode("utf-8")
print("Encoding utf-8:\n",st1)
st2 = st.encode("cp1251")
print("Encoding cp1251:\n",st2)
st3 = st2.decode("cp1251")
print("Decoding cp1251:\n",st3)
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Результат выполнения программы приведен ниже:

```
What is your group number?  P0001
Your group is  P0001

String:
P 0 0 0 1
String:
P 0 0 0 1

Hello, World!
Count 'o':  2
Find 'world' (by find()):  -1
Find 'World' (by find()):  7
Find 'World' (by index()):  7
Find 'World' (by in operation):  True

Replace substring:
 Old: Hello, World!
 New: hello, World!
```

```
List l1 (splitted):  ['Hello', ' World!']
List l2 (partitioned):  ('Hello', ',', ' World!')
Copy s in s2; s2:  Hello, World!
h|e|l|l|o


Encoding utf-8:
 b'\xd0\x9f\xd1\x80\xd0\xb8\xd0\xb2\xd0\xb5\xd1\x82,
\xd0\xbc\xd0\xb8\xd1\x80!'
Encoding cp1251:
 b'\xcf\xf0\xe8\xe2\xe5\xf2, \xec\xe8\xf0!'
Decoding cp1251:
 Привет, мир!
```

Строки

Строки в языке программирования Python представляют собой списки символов. Они могут быть объявлены тремя способами:

- с использованием одинарных кавычек: **'строка'**
- с использованием двойных кавычек: **"строка"**
- с использованием трех кавычек одинарного или двойного типа: **'''строка'''** или **"""строка"""**

Доступ к символам строки осуществляется аналогично доступу к элементам списка, используя позицию символа в строке. Длину строки можно получить с помощью функции **len(s)**, где **s** – строка. Копирование значений строк происходит с использованием оператора присваивания. Конкатенация строк осуществляется с помощью операции сложения:

```
s1 = s + "abc"
```

Для работы со строками могут быть использованы функции, описанные далее в таблице. В квадратных скобках указаны аргументы, которые могут быть опущены.

Функция	Описание	Пример
count(*substr*)	Подсчет количества повторов подстроки ***substr*** в строке	**num = s.count('o')**

find(*substr*)	Поиск подстроки в строке. Возвращает позицию первого вхождения подстроки ***substr*** в строку. Если подстрока ***substr*** не найдена, возвращает **-1**. Другим вариантом проверки вхождения подстроки ***substr*** в строку является запись: ***substr* in s**, где **s** – строка	**ind = s.find("ab")**
index(*substr*)	Поиск подстроки ***substr*** в строке. Возвращает позицию первого вхождения подстроки ***substr*** в строку. Если подстрока ***substr*** не найдена, возвращает ошибку.	**ind = s.index("ab")**
replace(*oldS*, *newS*)	Возвращает строку, для которой выполнена замена подстроки ***oldS*** в предыдущей строке на новую подстроку ***newS***	**s1 = s.replace("ab", "AB")**
split([*sep*])	Разделение строки на элементы списка по разделителю ***sep***, если он указан, или по пробелу в ином случае.	**l1 = s.split(",")**
partition(*sep*)	Разделение строки на элементы списка, состоящего из трех элементов – часть строки до разделителя, разделитель (***sep***), часть строки после разделителя.	**l2 = s.partition(",")**
join(*iterable*)	Формирование строки из любой итерируемой структуры данных ***iterable*** (может быть указан разделитель в качестве основной строки).	**s = ";".join(l)**
encode(*encoding*)	Возвращает новую строку, являющуюся копией первой в кодировке ***encoding***	**st1 = st.encode("utf-8")**

decode(*encoding*)	Возвращает новую строку, являющуюся декодированной копией первой из кодировки ***encoding***	**st1 = st.decode("utf-8")**

В таблице приведены основные функции. Со списком дополнительных функций можно ознакомиться в документации по языку программирования Python в разделе, посвященном стандартным типам данных:
https://docs.python.org/3/library/stdtypes.html#str

12. Дополните код программы **lab_02_04.py**. Создайте строковую переменную **season**, заполнив ее названием текущего сезона, введенным с клавиатуры. Смените кодировку данной строки на **utf-8**. Выведите закодированное и декодированное значения на экран.

13. Дополните код программы **lab_02_04.py**. Создайте переменную **sh** со значением *"When shall we three meet again; In thunder, lightning, or in rain?"*. Осуществите замену слов *"thunder"*, *"lightning"* и *"in rain"* на подстроки *"C"*, *"Erlang"* и *"Java main"* соответственно. Осуществите вывод итоговой строки на экран.

14. Создайте программу **lab_02_05.py**, которая выводит на экран все возможные уникальные строки, составленные из символов строки введенной с клавиатуры.

15. Создайте программу **lab_02_06.py**, которая выводит на экран дополнительный код введенного с клавиатуры шестнадцатеричного числа на восемь разрядов.

16. Создайте программу **lab_02_07.py**, которая преобразует введенное с клавиатуры двенадцатеричное число в систему с основанием 14 и выводит результат преобразования на экран.