Часть 1

Условные операторы: `if`, `elif`, `else`, операторы сравнения, каскады

Условные операторы позволяют программе принимать решения в зависимости от выполнения определенных условий.
Основные операторы:
- `if` — проверяет условие, и если оно истинно, выполняется блок кода внутри `if`.
- `elif` — используется для проверки дополнительных условий, если предыдущие условия не выполнились.
- `else` — выполняется, если ни одно из условий выше не было истинным.

Операторы сравнения:

- `==` — равно
- `!=` — не равно
- `>` — больше
- `<` — меньше
- `>=` — больше или равно
- `<=` — меньше или равно

Пример кода:

```
x = 10

if x > 10:
    print("x больше 10")
elif x == 10:
    print("x равно 10")
else:
    print("x меньше 10")
```

Вывод:

```
x равно 10
```

Так же при указании аргумента сравнения можно использовать операторы **and** и **or**, означающие «и» и «или» соответственно. And позволит выполниться if если выполняются оба условия, or - если хотя бы одно

Пример кода:

```
if a > c and b > c:
    do()
```

Часть 2

Циклы `for` и `while`

Циклы позволяют повторять выполнение блока кода несколько раз.
- `for` — используется для перебора элементов последовательности (например, списка или строки).
- `while` — выполняется, пока условие истинно.

Для запуска цикла посредством for используется конструкция

```
for переменная_счетчик in набор_объектов:
    Выполнять()
```

Где переменная_счетчик - переменная которая будет по очереди принимать значения из набор_объектов
Для задачи набора объектов можно использовать функцию range() или указать готовый список из объектов

Для работы функция range() принимает от 1 до 3 аргументов
range(start, stop, step)
Где
start - начало, включительно
stop - конец, не включительно
step - шаг, с которым будут меняться значения

если указать 1 аргумент - то start будет приниматься за 0, stop - то что будет указано, а step будет равен 1
Если указывать 2 аргумента, то будет изменяться start, а при указании 3го будет изменяться уже step

Пример кода:

```
# Цикл for
for i in range(3):
   print(«Итерация », i)

# Цикл while
counter = 0
while counter < 3:
   print(«Счетчик: » , counter)
   counter += 1
```

Вывод:

```
Итерация 0
```

Итерация 1
Итерация 2
Счетчик: 0
Счетчик: 1
Счетчик: 2

Часть 3

Строки, способы работы с ними, индексы

Строка - тип данных, позволяющий работать с символами и их последовательностями. Их можно складывать (‘qwe’ + ‘asd’ = ‘qweads’), умножать на целое (‘zxc’ * 2 = ‘zxczxc’), а так же индексировать и делать срезы
Для индексации необходимо в квадратных скобках без пробела после переменной, содержащей строку указать индекс необходимого символа. Для указания последнего символа можно использовать -1, предпоследнего - -2 и так далее.
Для срезания используются аналогичные функции range() переменные start, stop, step, однако указываются они через «:» все еще в квадратных скобках
В переменные start, stop, step подаются индексы, поэтому нельзя использовать те, которых нет в строке. Если пропустить какое-то из значений, то start будет заменен на 0, stop на длину строки, step на 1

Пример кода:
```
text = "Инжинириум"

# Индексация
print(text[0])  # Первый символ
print(text[-1]) # Последний символ

# Срезы
print(text[0:6])  # Символы с 0 по 5
print(text[8:])   # Символы с 8 до конца
```

Вывод:

```
И
м
Инжини
ум
```

функции и методы, применимые к строкам

Строки в Python имеют множество встроенных методов для работы с ними:
- `.len()` — длина строки.
- `.lower()` — преобразует строку в нижний регистр.
- `.isLower()` — проверяет, является ли строка вся в нижнем регистре
- `.upper()` — преобразует строку в нижний регистр.

- `.isUpper()` — проверяет, является ли строка вся в верхнем регистре
- `.replace()` — заменяет часть строки на другую.

Так же эти функции и методы можно применять к индексированным и срезанным частям, так как те так же являются строками

Пример кода:

```
text = «Будем писать ПРОВЕРОЧНУЮ"

print(len(text))
print(text.lower())
print(text.replace("ПРОВЕРОЧНУЮ", «анекдоты"))
```

Вывод:

```
24
будем писать проверочную
Будем писать анекдоты
```

Часть 4

Списки, функции и методы, применимые к ним

Список (list) в Python - это упорядоченный набор объектов, который может содержать элементы разных типов данных. Списки очень гибки и могут быть изменены (добавлены, удалены или изменены элементы)

К объектам списка можно обращаться через индексы, аналогично строкам, но в отличие от строк: выбранным элементом будет не отдельный символ, а какое-либо слово/число

Методы, применимые к спискам:
- `.append()` — добавляет элемент в конец списка. Метод
- `.remove()` — удаляет элемент по значению. Метод
- `.pop()` — удаляет элемент в конце списка. Метод
- `len()` — возвращает количество элементов в списке. Функция

Пример кода:

```
spisok = ["gta5", "dota2", "terraria"]


spisok.append("minecraft")
spisok.remove("gta5")

print(spisok)
print(len(spisok))
```

Вывод:

["dota2", "terratia", "minecraft"]
3

Часть 5

Объявление и вызов функций, локальные и глобальные переменные

Функция — это особым образом сгруппированный набор команд, которые выполняются последовательно, но воспринимаются как единое целое. Функция может возвращать (или не возвращать) результат.

Функция без возврата:

```
def cat():
   print("Мяу»)

cat()
```

Данная функция не будет ничего возвращать, так как тут не указан return, однако, так как одной из команд в ней является print(), то в консоль будет выведено

Мяу

Функция с возвратом:

```
def dog():
    return(‘Гав’)

dog()
```

Данная функция сама по себе ничего не выводит, так как она просто возвращает значение. Однако, если мы обернем ее в print()
print(dog())
Мы получим в консоли то, что возвращает данная функция, в данном случае

Гав

Использование функции:

Чтобы использовать функцию, её нужно сначала создать (объявить). Для этого используется ключевое слово def, за которым следует имя функции и круглые скобки.

Вызов функции — это обращение к ранее объявленной функции с целью выполнения её команд.

cat()

То есть чтобы наша функция заработала, необходимо ее запустить, без этого, ничего не будет происходить, ничего не будет выводиться в консоль

Функции с аргументами:

Так же в скобки при вызове функции можно подать какой-либо аргумент, после чего его можно будет использовать в функции, он будет храниться в локальной переменной (о ней позже)

Например:

```
def privet(name):
    print('Hello', name)

privet('Игорь')
```

Вывод:

Привет Игорь

Так же мы можем вводить несколько аргументов и работать с ними:

```
def slojit(a, b):
    s = a + b
    return s

print(slojit(5, 2))
```

Вывод:

7

Локальные и глобальные переменные:

В данном случае name будет являться локальной переменной, ее отличие от глобальной заключается в том, что локальная переменная может быть использована только внутри функции, в которой указана, а глобальная может быть использована где-угодно