еще. Комментарий помечается символом #; все, что находится после # до конца текущей строки, является комментарием. Обычно комментарий располагается на отдельной строке, как показано здесь:

```
>>> # 60 с/мин * 60 мин/ч * 24 ч/день
>>> seconds_per_day = 86400
```

Или на той же строке, что и код, который нужно пояснить:

```
>>> seconds_per_day = 86400 # 60 с/мин * 60 мин/ч * 24 ч/день
```

Символ # имеет много имен: хеш, шарп, фунт или устрашающее октоторп. Как бы вы его ни назвали, его эффект действует только до конца строки, на кото рой он располагается.

Python не дает возможности написать многострочный комментарий. Вам нужно явно начинать каждую строку или раздел комментария с символа #:

```
>>> # Я могу сказать здесь всё, даже если Python это не нравится,
>>> # поскольку я защищен крутым
>>> # октоторпом.
```

Однако если октоторп находится внутри текстовой строки, он становится простым символом #:

```
>>> print("Без комментариев: кавычки делают октоторп # безвредным")
Без комментариев: кавычки делают октоторп # безвредным
```

Оборудование и материалы.

Персональный компьютер, среда разработки Python.

Указания по технике безопасности:

Соответствуют технике безопасности по работе с компьютерной техникой.

Задания

***Продлеваем строки с помощью символа ***

Любая программа становится более удобочитаемой, если ее строки относительно короткие. Рекомендуемая (но не обязательная) максимальная длина строки равна 80 символам. Если вы не можете выразить свою мысль в рамках 80 символов, воспользуйтесь символом возобновления \. Просто поместите его в конце строки, и дальше Python будет действовать так, будто это все та же строка.

Например, если бы я хотел создать длинную строку из нескольких коротких, я мог бы сделать это пошагово:

```
>>> alphabet = ''
>>> alphabet += 'abcdefg'
>>> alphabet += 'hijklmnop'
>>> alphabet += 'qrstuv'
>>> alphabet += 'wxyz'
>>> alphabet
'abcdefghijklmnopqrstuvwxyz'
```

Или же за одно действие, использовав символ *continuation*:

```
>>> alphabet = ''
>>> alphabet = 'abcdefg' + \
           'hijklmnop' + \
           'qrstuv' + \
           'wxyz'
```

Продлить строку может быть необходимо, если выражение располагается на нескольких строках:

```
>>> 1 + 2 + \
  3
6
>>>
```

Сравниваем выражения с помощью операторов if, elif и else

До этого момента мы говорили только о структурах данных. Теперь же наконец готовы сделать первый шаг к рассмотрению структур кода, которые вводят данные в программы. (Вы уже могли получить представление о них в лабораторной работе 4, в разделе о множествах.) В качестве первого примера рассмотрим небольшую программу, которая проверяет значение булевой переменной disaster и выводит подходящий комментарий:

```
>>> disaster = True
>>> if disaster:
        print("Караул!")
else:
        print("Бугагашенька!")


Караул!
>>>
```

Строки if и else в Python являются операторами, которые проверяют, является ли значение выражения (в данном случае переменной disaster) равным True. Помните, print() — это встроенная в Python функция для вывода информации, как правило, на ваш экран.

> ***Примечание:***
>
> *Если вы работали с другими языками программирования, обратите внимание на то, что при проверке if вам не нужно ставить скобки. Не нужно писать что-то вроде if (disaster == True).*
>
> *В конце строки следует поставить двоеточие (:). Если вы, как и я, иногда забываете ставить двоеточие, Python выведет сообщение об ошибке.*

Каждая строка print() отделена пробелами под соответствующей проверкой. Здесь использовано четыре пробела для того, чтобы выделить каждый подраздел. Хотя вы можете использовать любое количество пробелов, Python ожидает, что внутри одного раздела будет применяться одинаковое количество пробелов. Рекомендованный стиль — PEP-8 (http://bit.ly/pep-8) — предписывает использовать четыре пробела. Не применяйте табуляцию или сочетание табуляций и пробелов — это мешает подсчитывать отступы.

Все выполненные в этом примере действия выполняют следующее:

1. Присвоили булево значение True переменной disaster.
2. Произвели условное сравнение с помощью операторов if и else, выполняя разные фрагменты кода в зависимости от значений переменной disaster.
3. Вызвали функцию print(), чтобы вывести текст на экран.

Можно организовывать проверку в проверке столько раз, сколько вам нужно:

```
>>> furry = True
>>> small = True
>>> if furry:
        if small:
                print("Это котэ")
        else:
                print("Это медведь")
else:
        if small:
                print("Это рыба")
        else:
                print("Это человек. Или лысый медведь.")


Это котэ
>>>
```

В Python отступы определяют, какие разделы if и else объединены в пару. Наша первая проверка обращалась к переменной furry. Поскольку ее значение равно True, Python переходит к выделенной таким же количеством пробелов проверке if small.

Поскольку мы указали значение переменной small равным True, проверка вернет результат True. Это заставит Python вывести на экран строку *Это котэ*.

Если необходимо проверить более двух вариантов, используйте операторы if, elif (это значит else if — «иначе если») и else:

```
>>> color = "Терракотовый"
>>> if color == "Красный":
        print("Это помидор")
elif color =="Зелёный":
        print("Это зелёный перец")
elif color == "Нежно-пурпурный":
        print("Таких овощей я не знаю")
else:
        print("Я никогда не слышал цвет ", color)


Я никогда не слышал цвет  Терракотовый
```

В предыдущем примере мы проверяли равенство с помощью оператора ==. В Python используются следующие операторы сравнения:

- равенство (==);
- неравенство (!=);
- меньше (<);
- меньше или равно (<=);
- больше (>);
- больше или равно (>=);
- включение (in …).

Эти операторы возвращают булевы значения True или False. Взглянем на то, как они работают, но сначала присвоим значение переменной x:

Выполним несколько проверок:

```
>>> x = 7
>>> x == 5
False
>>> x == 7
True
>>> 5 < x
True
>>> x < 10
True
```

Обратите внимание на то, что для проверки на равенство используются два знака «равно» (==); помните, что один знак «равно» применяется для присваивания значения переменной.

Если вам нужно выполнить несколько сравнений одновременно, можете использовать булевы операторы and, or и not, чтобы определить итоговый двоичный результат.

Булевы операторы имеют более низкий приоритет, нежели фрагменты кода, которые они сравнивают. Это значит, что сначала высчитывается результат фрагментов, а затем они сравниваются. В данном примере из-за того, что мы устанавливаем значение x равным 7, проверка 5 < x возвращает значение True и проверка x < 10 также возвращает True, поэтому наше выражение преобразуется в True and True:

```
>>> 5 < x and x < 10
True
```

Однако, самый простой способ избежать путаницы — использовать круглые скобки:

```
>>> (5 < x) and (x < 10)
True
```

Рассмотрим некоторые другие проверки:

```
>>> 5 < x or x < 10
True
>>> 5 < x and x > 10
False
>>> 5 < x and not x > 10
True
```

Если вы используете оператор and для того, чтобы объединить несколько проверок, Python позволит вам сделать следующее:

```
>>> 5 < x < 10
True
```

Это выражение аналогично проверкам 5 < x и x < 10. Вы также можете писать более длинные сравнения:

```
>>> 5 < x < 10 < 999
True
```

Что есть истина? Что, если элемент, который мы проверяем, не является булевым? Чем Python считает True и False?

Значение false не обязательно явно означает False. Например, к False приравниваются все следующие значения:

- булева переменная False;
- значение None;
- целое число 0;
- число с плавающей точкой 0.0;
- пустая строка (' ');
- пустой список ([]);
- пустой кортеж (());
- пустой словарь ({});
- пустое множество (set()).

Все остальные значения приравниваются к True. Программы, написанные на Python, используют это определение истинности (или, как в данном случае, ложности), чтобы выполнять проверку на пустоту структуры данных наряду с проверкой на равенство непосредственно значению False:

```
>>> some_list = []
>>> if some_list:
        print("Тут что-то есть!")
else:
        print("Тут пусто!")


Тут пусто!
```

Если вы выполняете проверку для выражения, а не для простой переменной, Python оценит его значение и вернет булев результат. Поэтому, если вы введете следующее:

```
>>> if color == "Красный":
```

Python оценит выражение color == "red". В нашем примере мы присвоили переменной color значение "Терракотовый", поэтому значение выражения color == "red" равно False и Python перейдет к следующей проверке:

```
elif color =="Зелёный":
```

Контрольные вопросы

1. Как можно продлить строку? Связать несколько строк в одну?
2. Какие операторы сравнения в Python вы знаете?
3. Какова структура оператора if?
4. Как записать в условии знак равенства?
5. Как записать в условии знак неравенства?
6. Как задать условие «меньше либо равно», «больше либо равно»?
7. Как в условии можно проверить включение?
8. Что возвращает оператор сравнения?
9. Как и для чего используются булевы операторы and, or, not?
10. Какой приоритет имеют булевы операторы по сравнению с операторами сравнения?
11. Что произойдет, если элемент, который проверяется в условии, не будет булевого типа?

Список литературы, рекомендуемый к использованию по данной теме:

1. Орлов, С. А. Программная инженерия. Технологии разработки программного обеспечения : учебник / С.А. Орлов. - 5-е изд., обновл. и доп. - СПб. : Питер, 2017. - 640 с.
2. Гагарина, Л. Г. Современные проблемы информатики и вычислительной техники : [учеб. пособие] / Л.Г. Гагарина, А.А. Петров. - М. : Форум, 2016. - 368 с.
3. Шкляр, М. Ф. Основы научных исследований : учеб. пособие / М.Ф. Шкляр. - 6-е изд. - М. : Дашков и Ко, 2016. - 208 с.
4. Михеева, Е. В. Информационные технологии в профессиональной деятельности : учеб. пособие / Е.В. Михеева. - 14-е изд., стер. - М. : Академия, 2016. - 384 с.
5. Любанович Билл. Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.

Лабораторная работа 6. Работа с циклами в Python

Цель работы:

Изучить команды создания циклов в языке Python. Изучить различия между циклами while и for и области их применения. Изучить применение команд break и continue.

Компетенции:

Код	Формулировка:
ОК-5	быть готовым работать с информацией в различных формах, использовать для ее получения, обработки, передачи, хранения и защиты современные компьютерные технологии
ОПК-5	способностью использовать основные приемы обработки и представления экспериментальных данных
ОПК-6	способность осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных,