Практическая работа №3. Условные операторы в Python.

Повторить и записать в тетрадь:

Типы данных в Python

В Python типы данных можно разделить на

- встроенные в интерпретатор *(built-in*) и
- не встроенные, которые можно использовать при импортировании соответствующих модулей.

К основным встроенным типам относятся:

1. *None* (неопределенное значение переменной)
2. Логические переменные (*Boolean Type*)
3. Числа (*Numeric Type*)
    1. *int* – целое число
    2. *float* – число с плавающей точкой
    3. *complex* – комплексное число
4. Списки (*Sequence Type*)
    1. *list* – список
    2. *tuple* – кортеж
    3. *range* – диапазон
5. Строки (*Text Sequence Type*)
    1. *str*
6. Бинарные списки (*Binary Sequence Types*)
    1. *bytes* – байты
    2. *bytearray* – массивы байт
    3. *memoryview* – специальные объекты для доступа к внутренним данным объекта через protocol buffer
7. Множества (*Set Types*)
    1. *set* – множество
    2. *frozenset* – неизменяемое множество
8. Словари (*Mapping Types*)
    1. *dict* – словарь

Что изучим?

оператор ветвления if и операторы цикла while и for.

Что мы знаем?

Условные операторы

Условный оператор ветвления *if*

Оператор ветвления *if* позволяет выполнить определенный набор инструкций в зависимости от некоторого условия. Возможны следующие варианты использования.

1. Конструкция *if*

Синтаксис оператора *if* выглядит так.

```
if выражение:
    инструкция_1
    инструкция_2
    ...
    инструкция_n
```

После оператора *if* записывается выражение. Если это выражение истинно, то выполняются инструкции, определяемые данным оператором. Выражение является истинным, если его результатом является число не равное нулю, непустой объект, либо логическое *True*. После выражения нужно поставить двоеточие ":".

ВАЖНО: блок кода, который необходимо выполнить, в случае истинности выражения, отделяется четырьмя пробелами слева!

Примеры:

```
if 1:
   print("hello 1")
```

Напечатает: *hello 1*

```
a = 3
if a == 3:
    print("Здесь a =", a)
```

Напечатает: Здесь a = 3

```
if a > 1:
   print("hello 3")
```

Напечатает: *hello 3*

```
lst = [1, 2, 3]
if lst :
   print("hello 4")
```

Напечатает: *hello 4*

2. Конструкция *if – else*

Бывают случаи, когда необходимо предусмотреть альтернативный вариант выполнения программы. Т.е. при истинном условии нужно выполнить один

набор инструкций, при ложном – другой. Для этого используется конструкция *if – else*.

```
if выражение:
    инструкция_1
    инструкция_2
    ...
    инструкция_n
else:
    инструкция_a
    инструкция_b
    ...
    инструкция_x
```

Примеры.

```
a = 3
if a > 2:
  print("H")
else:
  print("L")
```

Напечатает: *H*

```
a = 1
if a > 2:
  print("H")
else:
  print("L")
```

Напечатает: *L*

Условие такого вида можно записать в строчку, в таком случае оно будет представлять собой тернарное выражение.

```
a = 17
b = True if a > 10 else False
print(b)
```

В результате выполнения такого кода будет напечатано: *True*

3. Конструкция *if – elif – else*

Для реализации выбора из нескольких альтернатив можно использовать конструкцию *if – elif – else*.

```
if выражение_1:
    инструкции_(блок_1)
elif выражение_2:
    инструкции_(блок_2)
elif выражение_3:
    инструкции_(блок_3)
else:
    инструкции_(блок_4)
```

Пример.

```
a = int(input("введите число:"))
if a < 0:
   print("Neg")
elif a == 0:
   print("Zero")
else:
   print("Pos")
```

Если пользователь введет число меньше нуля, то будет напечатано “*Neg*“, равное нулю – “*Zero*“, большее нуля – “*Pos*“.

Практическая работа №4. Операторы цикла *while и* for

Оператор цикла *while* выполняет указанный набор инструкций до тех пор, пока условие цикла истинно. Истинность условия определяется также как и в операторе *if*. Синтаксис оператора *while* выглядит так.

```
while выражение:
    инструкция_1
    инструкция_2
    ...
    инструкция_n
```

Выполняемый набор инструкций называется телом цикла.Пример.

```
a = 0
while a < 7:
  print("A")
  a += 1
```

Буква “A” будет выведена семь раз в столбик.

Пример бесконечного цикла.

```
a = 0
while a == 0:
  print("A")
```

Операторы *break* и *continue*

При работе с циклами используются операторы ***break*** и ***continue***.

Оператор ***break*** предназначен для досрочного прерывания работы цикла *while*.

Пример.

```
a = 0
while a >= 0:
  if a == 7:
     break
  a += 1
  print("A")
```

В приведенном выше коде, выход из цикла произойдет при достижении переменной *a* значения 7. Если бы не было этого условия, то цикл выполнялся бы бесконечно.

Оператор ***continue*** запускает цикл заново, при этом код, расположенный после данного оператора, не выполняется.

Пример.

```
a = -1
while a < 10:
  a += 1
```

```
  if a >= 7:
      continue
  print("A")
```

При запуске данного кода символ “A” будет напечатан 7 раз, несмотря на то, что всего будет выполнено 11 проходов цикла.

Оператор цикла *for*

Оператор *for* выполняет указанный набор инструкций заданное количество раз, которое определяется количеством элементов в наборе.

Пример.

```
for i in range(5):
  print("Hello")
```

В результате *“Hello”* будет выведено пять раз.

Внутри тела цикла можно использовать операторы ***break*** и ***continue***, принцип работы их точно такой же как и в операторе *while*.

Если у вас есть заданный список, и вы хотите выполнить над каждым элементом определенную операцию (возвести в квадрат и напечатать получившееся число), то с помощью *for* такая задача решается так.

```
lst = [1, 3, 5, 7, 9]
for i in lst:
  print(i ** 2)
```

Также можно пройти по всем буквам в строке.

```
word_str = "Hello, world!"
for g in word_str:
  print(g)
```

Строка *“Hello, world!”* будет напечатана в столбик.

Задание:

1) **Отработать Пайтоне все методы по теме**
2) **Залить код в репозиторий**