Тема занятия: Встроенные типы данных и операции с ними

1. Переменные и типы данных

Переменная — это простейшая именованная структура данных, в которой может быть сохранен промежуточный или конечный результат работы программы. Объект (в Python все является объектом), на который ссылается переменная, называется значением переменной.

К основным встроенным типам относятся:

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

Присвоение переменной – процесс указания переменной объекта, на который она будет ссылаться. В Python присвоение осуществляется с помощью =.

Оператор или инструкция – наименьшая часть языка программирования. Это команда или набор команд, которые могут быть

выполнены исполнителем. Объекты, над которыми операторы производят действия, называются операндами.

Виды операторов:

- Унарные. Один операнд
- Бинарные. Два операнда
- Тернарные. Три Операнда

Выражение – совокупность переменных, операторов, функций, которую можно вычислить в соответствии с синтаксисом языка.

Обязательные атрибуты объекта:

- идентификатор
- значение
- тип

Функция id(<переменная>) нужна для того, чтобы посмотреть на объект с каким идентификатором ссылается данная переменная.

К неизменяемым относятся

- целые числа (int)
- числа с плавающей точкой (float)
- комплексные числа (complex)
- логические переменные (bool)
- кортежи (tuple)
- строки (str)
- неизменяемые множества (frozen set)

К изменяемым относятся

- списки (list)
- множества (set)
- словари (dict)

Неизменяемость типа данных означает, что созданный объект больше не изменяется.

2. Ввод и вывод информации

Функция input([<str>]) нужна для получения данных от пользователя. Данная функция позволяет перед запросом ввода вывести текст.

Функция print([<переменная>, …]) нужна для отображения данных на экране. С её помощью можно выводить строки, числа, последовательности.

3. Арифметические операторы

Оператор (функция)	Описание	Пример

- (бинарный)	Вычитание.	`one = 1`
		`two = 2`
		`print(one - two)  # -1`
		`print(one - 2)  # -1`
		`print(two - one)  # 1`
		`print(1 - 2)  # -1`

+	Сложение.	`one = 1`
		`two = 2`
		`print(one + two)  # 3`
		`print(one + 2)  # 3`
		`print(1 + 2)  # 3`

*	Умножение.	`one = 1`
		`two = 2`
		`print(one * two)  # 2`
		`print(one * 2)  # 2`
		`print(1 * 2)  # 2`

/	Деление. Результат деления всегд float.	`one = 1`
		`two = 2`
		`print(one / two)  # 0.5`
		`print(one / 2)  # 0.5`
		`print(two / one)  # 1.0`
		`print(1 / 2)  # 0.5`

**	Возведение в степень.	`five = 5` `two = 2`

		`print(five ** two)# 25`
		`print(two ** five)# 32`
		`print(5 ** two)  # 25`

- (унарный)	Унарный минус.	`nine = 9`
		`minus_nine = -nine`
		`print(minus_nine)  # -9`

//	Целочисленное деление.
	Некоторые правила:
	1) Результат имеет тот же знак, что и при обычном делении.
	2) Следует, учитывая первый пункт, найти ближайшее к делимому меньшее число, которое нацело делится на делитель и разделить его на делитель, чтобы узнать результат.
		`six = 6`
		`four = 4`
		`print(six // four) # 1`
		`print(four // six) # 0`
		`print(six // -four)# -2`
		`print(-six // -four)# 1`
		`print(-six // four)# -2`

%	Остаток от деления.
	Некоторые правила:
	1) Остаток имеет тот же знак, что и делитель.
	2) Чтобы найти остаток, сперва нужно найти целое.
		`ten = 10`
		`three = 3`
		`print(ten % three) # 1`
		`print(-ten % three) # 2`
		`print(ten % -three)# -2`
		`print(-ten % -three)#-1`

abs	Модуль числа	`eight = 8`
		`print(abs(eight))  # 8`
		`print(abs(-eight)) # 8`

round	Округление. Вторым параметром можно передать количество знаков после запятой, до которых нужно округлять. Если второй параметр не передан округляет до целого.
	Если после знака, до которого нужно округлить стоит 5, то округляет до ближайшего четного.
	
	У Python проблемы с операциями с дробными числами, результат может отличаться от математического. Для точных вычислений лучше использовать Decimal.
		`first_n = 3.0001`
		`second_n = 2`
		`third_n = first_n / second_n`
		`print(round(third_n))`
		`# 2`
		`print(round(third_n, 4)) # 1.5001`
		
		`print(round(2.545, 2))`
		`# 2.54`
		`print(round(2.555, 2))`
		`# 2.56`

Арифметические операции с присвоением позволяют присвоить результат операции первому операнду.

4. Приведение типов

Функция type([<переменная>]) нужна для определения типа.

Преобразование типа – это процесс преобразования значения одного типа данных (целые числа, строки, числа с плавающей точкой и т. д.) в другой.

Виды преобразований:

- *Неявное преобразование типов.* При неявном преобразовании типов Python автоматически преобразует один тип данных в другой. Этот процесс не требует участия пользователя. Например, если операндами являются и float и int, то результат всегда будет иметь тип float.
- *Явное приведение типов.* В явном преобразовании программист сам заменяет текущий тип данных объекта на требуемый. Для этого используются встроенные функции, такие как int (), float (), str() и т. д., чтобы выполнить явное преобразование типов.

5. Операторы сравнения, логические операторы, операторы тождественности, операторы вхождения и побитовые операторы

Операторы сравнения:

Оператор (функция)	Описание	Пример
==	Проверяет равны ли значения обоих операндов. Если да, то условие становится истинным.	`print(3 == 3) # True` `print(4 == 5) # False`
!=	Проверяет равны ли оба значения обоих операндов. Если нет, то условие становится истинным.	`print(3 != 3) # False` `print(4 != 5) # True`

>	Проверяет больше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`print(6 > 3) # True` `print(6 > 6) # False`

<	Проверяет меньше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`print(6 < 3) # False`
		`print(6 < 6) # False`
		`print(6 < 7) # True`

>=	Проверяет больше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`print(6 >= 3) # True`
		`print(6 >= 6) # True`
		`print(6 >= 7) # False`

<=	Проверяет меньше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`print(6 <= 3) # False`
		`print(6 <= 6) # True`
		`print(6 <= 7) # True`

Логические операторы:

Оператор (функция)	Описание	Пример

and	Логический оператор "И". Условие будет истинным если оба операнда истина.	`print(True and True)`
		`# True`
		`print(True and False)`
		`# False`
		`print(6 < 7 and 'python' == 'python')`
		`# True`

or	Логический оператор "ИЛИ". Если хотя бы один из операндов истинный, то и все выражение будет истинным.	`print(True or True)`
		`# True`
		`print(True or False)`
		`# True`
		`print(False or False)`
		`# False`
		`print(6 < 7 or 'python' == 'python')`
		`# True`
		`print(6 > 7 or 'python' == 'python')`

		`# True`
		`print(6 > 7 or 'python' != 'python')`
		`# False`

not	Логический оператор "НЕ". Изменяет логическое значение операнда на противоположное.	`print(not True) # False` `print(not False) # True`

Операторы тождественности:

Оператор (функция)	Описание	Пример

is	Возвращает истину, если оба операнда указывают на один объект.	`number_1 = 5`
		`number_2 = 5`
		
		`print(id(number_1))`
		`print(id(number_2))`
		`# id одинаковые`
		`print(number_1 is number_2) # True`
		`print(number_1 is not number_2) # False`
		
		`number_1 = 389`
		`number_2 = 389`
		
		`print(id(number_1))`
		`print(id(number_2))`
		`# id разные`
		`print(number_1 is number_2) # False`
		`print(number_1 is not number_2) # True`

is not	Возвращает ложь если оба операнда указывают на один объект.	

Побитовые операторы:

Оператор (функция)	Описание	Пример

&	Бинарный "И" оператор, копирует бит в результат только если бит присутствует в обоих операндах.	`number_1 = 13`
		`# 0b00001101`
		`number_2 = 60`
		`# 0b00111100`
		`print(number_1 & number_2) # 12`

|	Бинарный "ИЛИ" оператор копирует бит, если тот присутствует в хотя бы в одном операнде.	`number_1 = 13`
		`# 0b00001101`
		`number_2 = 60`
		`# 0b00111100`
		`print(number_1 | number_2) # 61`

^	Бинарный "Исключительное ИЛИ" оператор копирует бит только если бит присутствует в одном из операндов, но не в обоих сразу.	`number_1 = 13`
		`# 0b00001101`
		`number_2 = 60`
		`# 0b00111100`
		`print(number_1 ^ number_2) # 49`

~	Бинарный комплиментарный оператор. Является унарным (то есть ему нужен только один операнд) меняет биты на обратные, там где была единица становиться ноль и наоборот.
	~n = -(n + 1)
		`number_1 = 13`
		`# 0b00001101`
		`number_2 = 60`
		`# 0b00111100`
		`print(~number_1) # -14`
		`print(~number_2) # -61`

<<	Побитовый сдвиг влево. Значение левого операнда "сдвигается" влево на количество бит указанных в правом операнде.
	n << k = n * (2**k)
		`number_1 = 13`
		`# 0b00001101`
		`print(number_1 << 1)`
		`# 26`

>>	Побитовый сдвиг вправо. Значение левого операнда "сдвигается" вправо на количество бит указанных в правом операнде.	`number_2 = 60`
		`# 0b00111100`
		`print(number_2 >> 2)`
		`# 15`

	n >> k = n / (2**k)	

Приоритет операторов (1 - наивысший):

1. **
2. ~ + -
3. * / % //
4. + -
5. >> <<
6. &
7. ^ |
8. <= < > >=
9. == !=
10. = %= /= //= -= += = *=
11. is is not
12. in not in
13. not or and

6. Условные операторы

Оператор ветвления if

Оператор ветвления if позволяет выполнить определенный набор инструкций в зависимости от некоторого условия.

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 или одним табом слева

Синтаксис оператора ветвления if:

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

Оператор ветвления 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.

Синтаксис оператора ветвления if-else:

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

Оператор ветвления if-elif-else

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

Синтаксис оператора ветвления 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)
```

7. Комментарии

Для того, чтобы сформировать комментарий, необходимо начать строку с #. Все, что находится справа от #, воспринимается как комментарий.