Ввод и вывод данных

Переменные и типы данных

Переменной в программировании называется именованный контейнер для некоторого значения. Представьте себе ящик с наклееным на него стикером. Вы кладете что-нибудь в ящик и пишете на стикере короткое имя без пробелов и знаков препинания, начинающееся с буквы английского алфавита, примерно так же работает переменная в программировании.

```
a = 10
A = 20
```

a и A - это разные переменные, регистр ввода имеет значение

Типы данных

Информация, получаемая нами с помощью различных органов чувств, делится на зрительную, слуховую, обонятельную, осязательную и другие. В программировании так же есть свое разделение, разделение на типы данных. Примитивных типов данных в Python:

`int` - целочисленный тип (1, 192, 781287)

`float` - вещественный (дробный) (1.12312, 1231.12378718)

`str` - строковый тип (обязательно пишется в кавычках) `('Hello world')`

`bool` - булевый тип (имеет всего два значения: `True` и `False`)

Приведение типов

Приведением типов данных называется преобразование одного типа в другой, например, строку в число, число в строку, число в булеву переменную, строку в дробь и так далее.

```
a = 10
b = str(a) # b - это строка
c = int(b) # c - это снова число
d = 10.78
e = int(d) # e равно 10
```

Функция print

Функция `print` выводит переданные в неё аргументы в стандартный поток вывода. Что же такое стандартный поток вывода? `Standart output` или `stdout` называется потоком вывода, местом, куда мы выводим наш текстовый контент. По

умолчанию стандартный поток вывода равен `sys.stdout` и поэтому вывод осуществляется в консоль.

Функция `print` все переданные в неё аргументы в стандартный поток вывода. Например:

```
print(1)
print('Hello world!')
print(False)
print(1.5, 2.0, 10, True, 'username')
print('Hello' + ' ' + 'world' + '!')
# 1
# Hello world!
# False
# 1.5 2.0 10 True username
# Hello world!
```

На этом тривиальном примере мы видим две вещи. Во-первых, каждый `print` переносит поток вывода на новую строчку. Во-вторых, аргументы, переданные в функцию `print` выводятся через пробел.

```
# Во-первых:
print(1)
print(2)
# 1
# 2
# Во-вторых:
print('Hello', 'world')
# Hello world
```

В обоих случаях мы можем изменить стандартное поведение. Рассмотрим первый параметр функции `print — end,` в него передается строка, которая напечатается после всех аргументов функции `print.`

```
print(1, end=' ')
print(2, end=' ')
print(3, end=' ')
# 1 2 3

print(1, end=''-'')
print(2, end='-')
print(3, end=''-'')
# 1-2-3-

print(1, end='-я выведусь после первого print-')
print(2, end='-а я после второго-')
print(3, end='-я выведусь после третьего-')
# 1-я выведусь после первого print-2-а я после второго-3-я
выведусь после третьего-
```

Рассмотрим второй параметр функции print — sep, sep от английского `separator (разделитель).` По умолчанию параметр `sep` равен `' '.`

```
print(1, 2, 3, 4)
print(1, 2, 3, 4, sep='+++')
print(1, 2, 3, 4, sep='разделитель')
print(1, 2, 3, 4, sep='(◕‿◕)')
print(1, 2, 3, 4, sep='(/◕ヮ◕)/*:･ﾟ✧')
# 1 2 3 4
# 1+++2+++3+++4
# 1разделитель2разделитель3разделитель4
# 1(◕‿◕)2(◕‿◕)3(◕‿◕)4
# 1(/◕ヮ◕)/*:･ﾟ✧2(/◕ヮ◕)/*:･ﾟ✧3(/◕ヮ◕)/*:･ﾟ✧4
```

Функция input

Функция `input` является функцией стандартного ввода `(stdin)`. Ее главная задача - это передача введенных пользователем данных в функцию.

```
name = input()
print('Hello, ' + name)
```

Функция `input` может принимать всего лишь один аргумент - строку, которая выведется перед кареткой ввода

```
name = input('Enter your name: ')
print('Hello, ' + name)
```

Функция `input` возвращает строковый тип данных

Строки можно складывать друг с другом - это называется конкатенацией или объединением

```
number = input()
print(type(number))
#
```

Поэтому если мы напишем такой код, то он будет работать некорректно:

```
number1 = input()
number2 = input()
print(number1 + number2)
# Ввод:
# 1
# 2
# Вывод:
# 12
```

Поэтому необходимо преобразовать строковый тип в целочисленный `(str в int)`

```
number1 = int(input())
number2 = int(input())
print(number1 + number2)
# Ввод:
```

```
# 1
# 2
# Вывод:
# 3
```

Всегда проверяйте тип полученных данных, это поможет вам избежать большого количества ошибок. К слову, если вы что-то напутали с типами переменных, то Python выдаст ошибку TypeError (ошибка типа)

Операторы в Python

В языке программирования Python есть арифметические, логические и операторы сравнения.

Сложение:

```
# Сложение
print(10 + 30)

# Вывод
>>
```

Вычитание:

```
# Вычитание
print(30 - 20)

# Вывод
>>
```

Умножение:

```
# Умножение
print(2 * 9)

# Вывод
>>
```

Деление:

```
# Деление
print(100 / 25)

# Вывод
>>
```

Кроме всем знакомой четверки есть и несколько экзотических операторов: взятие остатка от деления, деление нацело, возведение в степень.

Взятие остатка от деления:

```
# Взятие остатка от деления на 2
print(0 % 2)
print(1 % 2)
print(2 % 2)
print(3 % 2)
print(4 % 2)

# Вывод
>> 0 # 0 % 2
>> 1 # 1 % 2
>> 0 # 2 % 2
>> 1 # 3 % 2
>> 0 # 4 % 2

# Взятие остатка от деления на 3
print(0 % 3)
print(1 % 3)
print(2 % 3)
print(3 % 3)
print(4 % 3)
print(5 % 3)
print(6 % 3)

# Вывод
>> 0 # 0 % 3
>> 1 # 1 % 3
>> 2 # 2 % 3
>> 0 # 3 % 3
>>   # 4 % 3
>>   # 5 % 3
>>   # 6 % 3
```

Деление нацело:

```
# Деление нацело на 10
print(91 // 10)
print(85 // 10)
print(16 // 10)
print(8 // 10)

# Вывод
>> 9 # 91 // 10
>> 8 # 85 // 10
>> 1 # 16 // 10
>> 0 # 8 // 10

# Деление нацело на 2
print(14 // 2)
print(15 // 2)
print(7 // 2)
print 6 // 2

# Вывод
```

```
>> 7 # 14 // 2
>> 7 # 15 // 2
>> 3 # 7 // 2
>> 3 # 6 // 2
```

Возведение в степень:

```
# Возведение в степень числа 2
print(2 ** 0) # 1
print(2 ** 1) # 2
print(2 ** 2) # 2 * 2
print(2 ** 3) # 2 * 2 * 2
print(2 ** 4) # 2 * 2 * 2 * 2

# Вывод
>> 1
>> 2
>> 4
>> 8
>>
```

Операторы сравнения могут возвращать всего два результата: `True` и `False`.

Оператор равенства `==` возвращает `True`, если числа равны, и `False` в противном случае.

```
a = 10
b = 10
print(a == b)

# Вывод
>> True

a = 8
b = 7
print(a == b)

# Вывод
>>
```

Оператор неравенства `!=` возвращает `True`, если числа не равны, и `False` в противном случае.

```
a = 8
b = 7
print(a != b)

# Вывод
>>
```

Оператор больше > возвращает `True`, если первое число больше второго, и `False` в противном случае.

```
a = 8
b = 7
print(a > b)
print(b > a)

# Вывод
>> True # a > b
>>       # b > a
```

Оператор меньше < возвращает `True`, если первое число меньше второго, и `False` в противном случае.

```
c = 100
d = 200
print(c < d)
print(d < c)

# Вывод
>> True # c < d
>>       # d < c
```

Оператор меньше или равно `<=` возвращает `True`, если первое число меньше второго или равно ему, и `False` в противном случае.

Оператор больше или равно `>=` возвращает `True`, если первое число больше второго или равно ему, и `False` в противном случае.

```
c = 200
d = 200
print(c >= d)
print(d <= c)

# Вывод
>> True # c >= d
>>      # d <= c
```

Иногда требуются выполнение нескольких операторов сравнения сразу. Для таких целей существует оператор `and` (оператор логического умножения, конъюнкция).

```
print(10 > 0 and 5 > 0)
print(10 % 2 == 0 and 12 % 2 == 0) # оба числа четные

# Вывод:
>> True
>>
```

Если хотя бы один из операторов равен `False`, то результат оператора будет равен `False`. Конъюкция истинна в том случае, когда все условия истинны.

```
print(10 > 100 and 5 > 0 and 10 > 0) # False

# Вывод:
>>
```

Логическое сложение (дизъюнкция) или оператор `or` требует выполнения ХОТЯ БЫ одного условия.

```
print(10 > 100 or 5 > 100 or 10 > 0) # True
print(1 == 0 or 2 == 0) # False, оба условия ложны

# Вывод:
>> True
>>
```

Последний из операторов - это оператор инверсии `not`. Оператор `not` изменяет (инвертирует) значение на противоположное.

```
print(not False) # True
print(not True) # False
print(not 2 == 0) # True

# Вывод:
>> True
>> False
>>
```

Оператор `not` выполняется в приоритете.

```
print(not 1 == 0 or 2 == 0) # True, значение первого условия
инвертировано

# Вывод:
>>
```

Задания для выполнения:

Решение всех задач реализовать в одном файле. Для выбора решенной задачи использовать аргументы командной строки с использованием библиотеки **argparse** - Анализатор параметров командной строки, аргументов и подкоманд [https://docs.python.org/3/library/argparse.html]

Пример использования:

```
import argparse

parser = argparse.ArgumentParser(description = "Practica_day_1")
parser.add_argument('--number_task', default=1, type=int, help='Номер задачи')

opt = parser.parse_args()
number_task = opt.number_task

# Решения задач:
if number_task == 1:
	# Решение задачи 1
elif number_task == 2:
	# Решение задачи 2
```

```
elif number_task == 3:
      # Решение задачи 3

…
else number_task == n:              # n – номер последней задачи.
      # Решение задачи под номером n
```

1. Вывести строку «Привет, %Имя Фамилия%. Это мой %номер дня% день учебной практики»
   На месте слов с % должны быть введены данные с использованием input
2. В листинге программы ввести два числа, одно из которых целое, другое – дробное.
   Два числа (целое и дробное) ввести с использованием input.
   Выполнить сложение дробных чисел, и вычитание целых.
3. Даны числа n и m. n – количество букв в ваших имени и фамилии, m - количество букв в имени. Вывести следующие строки:
   m чисел предшествующие числу n это …, …
   m чисел следующих за числом n это …, …
4. Ввести имя и свой год рождения. Вывести строку:
   «Привет, %имя%! Сейчас Вам ... лет. Когда Вы закончите университет, Ваш возраст будет равен … .
5. Ввести два числа, проверить истинность выражения (по вариантам – вариант – порядковый номер в журнале):
   Четные варианты: «Числа имеют одинаковую четность»
   Нечетные варианты: «Числа имеют разную четность»
   Вывести True или False
6. Ввести три числа. Проверьте истинность выражения «Хотя бы одно из введенных чисел - положительное»
7. Ввести шестизначное число.
   Четные варианты: Вывести первую цифру числа
   Нечетные варианты: Вывести последнюю цифру числа
8. Найти сумму цифр трехзначного числа.
9. Ввести четырехзначное число. Проверить истинность высказывания: «Все цифры введенного числа - различны»
10. Ввести время (Часы и минуты). Посчитать сколько секунд прошло с начала суток.