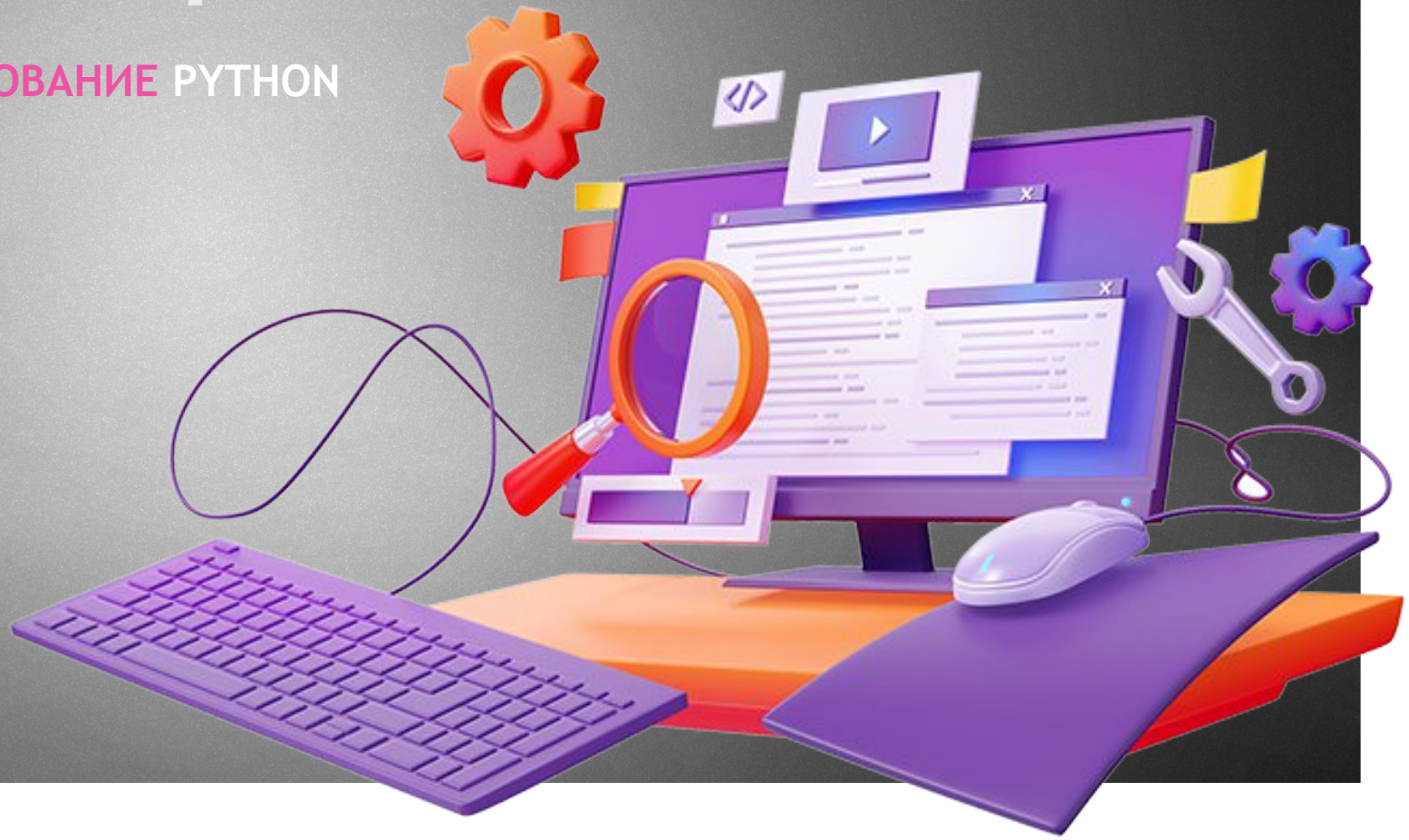
Программирование для начинающих

ВВЕДЕНИЕ В ЯЗЫК ПРОГРАММИРОВАНИЕ PYTHON



ПЛАН ВЕБИНАРА

- О языке программирования Python (особенности языка)
- Отличия языка от других языков и его преимущества
- Дзен Python
- Установка IDLE, Pycharm, онлайн версии
- Функции в Python
- Математические операции

О языке программирования Python

- **20.02.1991 г.** Автор языка – **Гвидо ван Россум** (Нидерланды)

- **Питон или Пайтон**

- в честь комедийного шоу **«Воздушный цирк Монти Пайтона»**

Особенности языка Python

- Высокоуровневый язык программирования
- Объектно-ориентированный
- Выделение блоков кода отступами
- Синтаксис ядра языка минималистичен

Преимущества среди других языков

- Наличие большого количества локализованных **учебных пособий**, доступных **онлайн**.
- Наличие среды программирования IDLE для операционных систем: **Windows**, **Linux** и **MacOS**.
- **Open Source** — у интерпретатора **Python** открытый код.
- Расширяемость и гибкость — **Python** можно легко расширить для взаимодействия с другими программными системами или встроить в программы в качестве компонента.
- Имеет более **читабельную структуру** кода.
- Не содержит синтаксических правил.
- Легко запоминается.

График популярности языков программирования

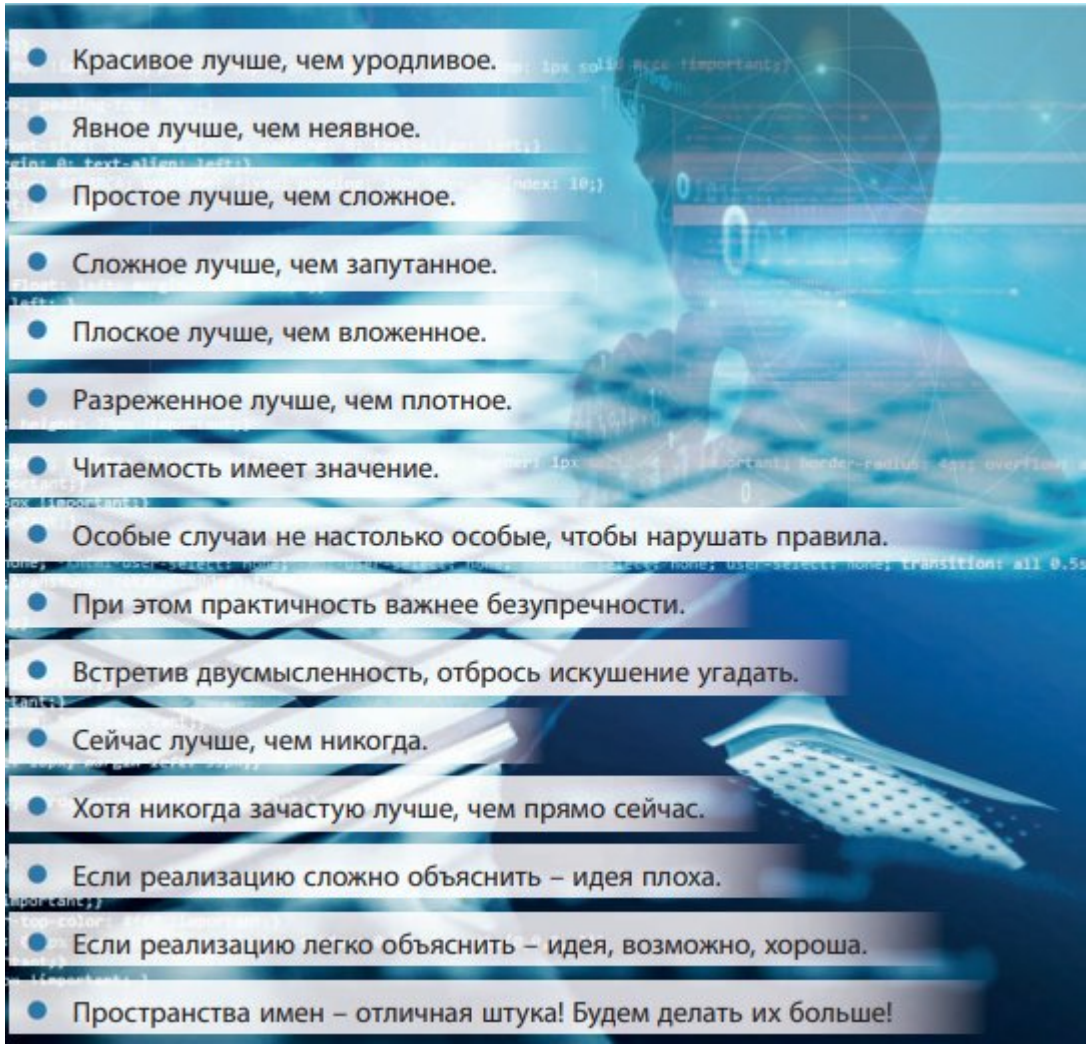
Dec 2021	Dec 2020	Change	Programming Language	Ratings	Change
1	3	↑	Python	12.90%	+0.69%
2	1	↓	C	11.80%	-4.69%
3	2	↓	Java	10.12%	-2.41%
4	4		C++	7.73%	+0.82%
5	5		C#	6.40%	+2.21%
6	6		Visual Basic	5.40%	+1.48%
7	7		JavaScript	2.30%	-0.06%
8	12	⇈	Assembly language	2.25%	+0.91%
9	10	↑	SQL	1.79%	+0.26%
10	13	↑	Swift	1.76%	+0.54%
11	9	↓	R	1.58%	-0.01%
12	8	⇊	PHP	1.50%	-0.62%

Дзен Python

- **Дзен Пайтона** — это философия программирования.

- В интерпретаторе **Python** он представлен как пасхальное яйцо: если ввести в команду **import this**, то в окне отобразится его текст

```
>>> import this
```

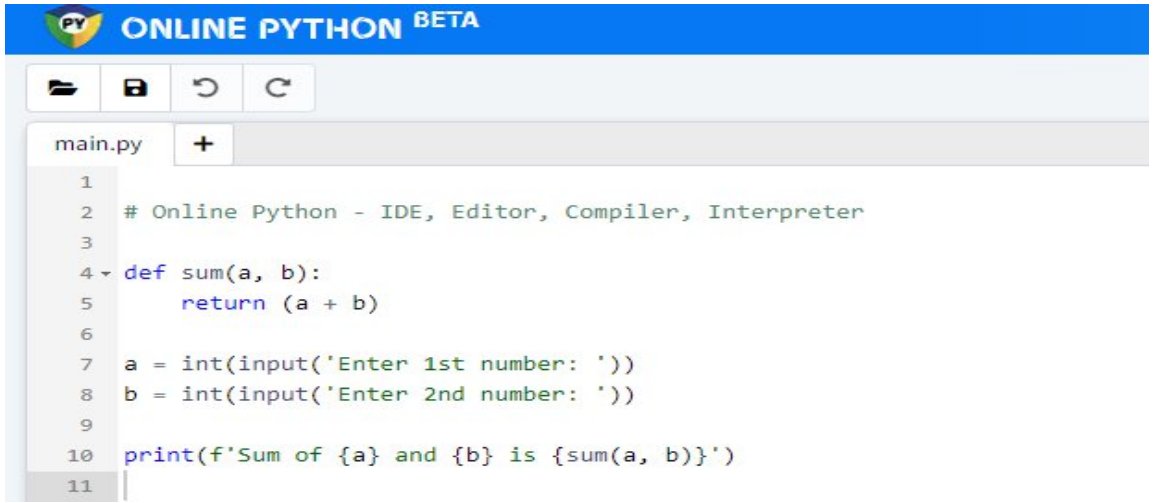


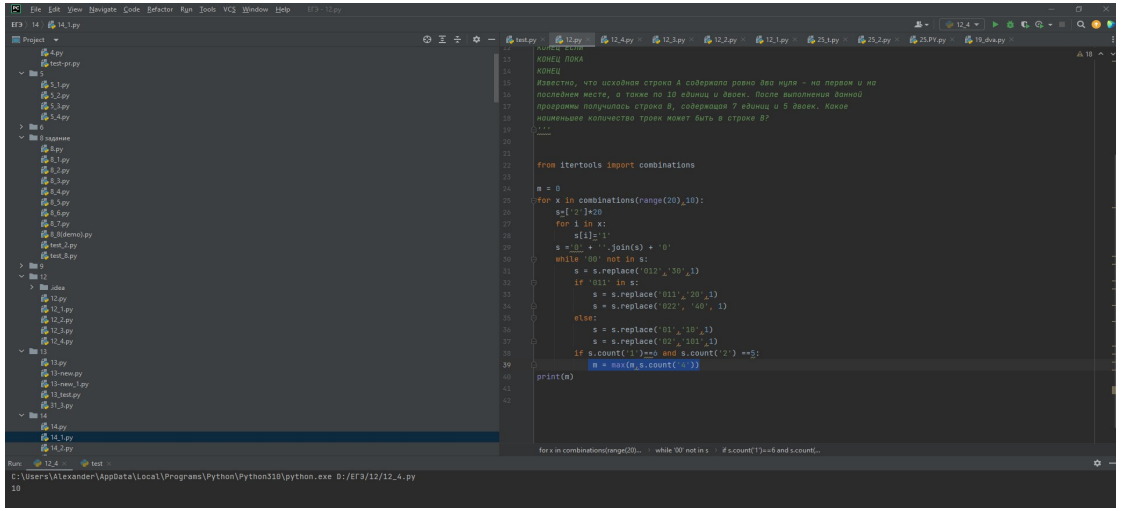

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

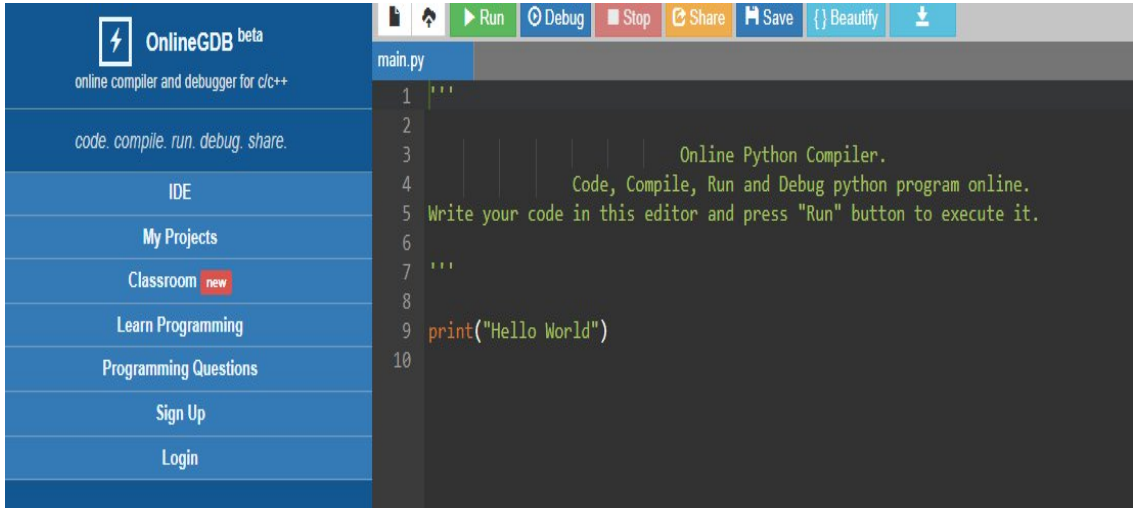
Программы

- Интегрированная среда IDLE
- Онлайн-среда (Repl.it)
- Pycharm


Python 3.4.3 Shell
File Edit Shell Debug Options Window Help
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 24 2015, 22:43:06) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> import numpy as np
>>> import scipy as sp
>>> import matplotlib as mp
>>>


ONLINE PYTHON BETA
main.py
# Online Python - IDE, Editor, Compiler, Interpreter
def sum(a, b):
return (a + b)
a = int(input('Enter 1st number: '))
b = int(input('Enter 2nd number: '))
print(f'Sum of {a} and {b} is {sum(a, b)}')




OnlineGDB beta
online compiler and debugger for c/c++
code. compile. run. debug. share.
IDE
My Projects
Classroom new
Learn Programming
Programming Questions
Sign Up
Login
Run
Debug
Stop
Share
Save
{} Beautify
main.py
'''
Online Python Compiler.
Code, Compile, Run and Debug python program online.
Write your code in this editor and press "Run" button to execute it.
'''
print("Hello World")

Для чего нужен PyCharm

- **Код писать удобнее.**
- **Возможности среды шире.**
- **Командная работа проще.**

Что позволяет делать PyCharm

- Создавать проекты.
- Корректировать.
- Писать код на Python.
- Устанавливать библиотеки и фреймворки.
- Запускать.
- Тестировать.
- Отлаживать.
- Устанавливать дополнения и плагины.
- Писать на других языках.

Полезные особенности PyCharm

- Использование шаблонов
- Автогенерация кода
- Встроенные инструменты
- Умный редактор
- Рефакторинг
- Умный поиск

Функции в Python

```
def my_function():
    print('hello from python')
```

```
my_function()
```

```
hello from python
```

Примеры Функций

- **abs()** -возвращает абсолютное значение числа

```
abs(-55)
```

```
55
```

- **cnr()** -возвращает строку, представляющую символ Unicode для переданного числа

```
chr(103)
```

```
'g'
```

Примеры Функций

- **callable()** - Функция callable() сообщает, является ли объект вызываемым. Если да, то возвращает True, а в противном случае — False.

```
callable(15)
```

```
False
```

- **complex()** - комплексное число, число, представленное в форме a + bi. Оно принимает целые числа или строки и возвращает соответствующее комплексное число.

```
complex(13)
```

```
(13+0j)
```

Примеры Функций

- **dict() -** Эта функция используется в Python для создания словарей. Это же можно делать и вручную, но функция предоставляет большую гибкость и дополнительные возможности. Например, ей в качестве параметра можно передать несколько словарей, объединив их в один большой.

```
dict({"a":1, "b":2}, c = 3)
```

```
{'a': 1, 'b': 2, 'c': 3}
```

Примеры Функций

- **dir() -** получает список вех атрибутов и методов объекта. Если объект не передать, то функция вернет все имена модулей в локальном пространстве имен.

```
x = ["Паскаль", "Пайтон", "C++"]
```

```
print(dir(x))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__',
'__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rm
ul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Примеры Функций

- **enumerate() -** В качестве параметра эта функция принимает последовательность. После этого она перебирает каждый элемент и возвращает его вместе со счетчиком в виде перечисляемого объекта. Основная особенность таких объектов — возможность размещать их в цикле для перебора.

```
x = 'Python'
list(enumerate(x))
```

```
[(0, 'P'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]
```

Примеры Функций

- **eval() -** Обрабатывает переданное в нее выражение и исполняет его как выражение Python.

```
eval('2*7')
```

```
14
```

```
eval('2+8')
```

```
10
```

```
eval('5//2')
```

```
2
```

Примеры Функций

- **filter()** - Как можно догадаться по названию, эта функция используется для перебора итерируемых объектов и последовательностей, таких как списки, кортежи и словари.

```
list1 = [6, 5, 8, 8, 6, 33, 22, 18, 76, 1]
result = list(filter(lambda x: (x%2 != 0) , list1))
print(result)
```

```
[5, 33, 1]
```

- **float()** - Эта встроенная функция конвертирует число или строку в число с плавающей точкой и возвращает результат.

```
float(578)
```

```
578.0
```

```
float(10)
```

```
10.0
```

Примеры Функций

- **hash() -** У большинства объектов в Python есть хэш-номер. Функция hash() возвращает значение хэша переданного объекта. Объекты с __hash__() — это те, у которых есть соответствующее значение.

```
hash('Hello World')
```

```
-3835318355628315070
```

Примеры Функций

- **help() -** Предоставляет простой способ получения доступа к документации Python без интернета для любой функции.

```
help(print)
```

```
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Примеры Функций

- **input() -** Это быстрый и удобный способ получить данные от пользователя. Вызов этой функции предоставляет пользователю возможность ввести на экране текст. Затем он конвертируется в строку и возвращается в программу.

```
x = input("Пожалуйста, введите значение: ")
print(x)
```

```
Пожалуйста, введите значение: 1234
1234
```

```
print(type(x))
```

```
<class 'str'>
```

- **int() -** Эта функция возвращает целое число из объекта, переданного в параметры. Она может конвертировать числа с разным основанием (шестнадцатеричные, двоичные и так далее) в целые.

```
x = int(input("Пожалуйста, введите значение: "))
print(x)
```

```
Пожалуйста, введите значение: 1234
1234
```

```
print(type(x))
```

```
<class 'int'>
```

Примеры Функций

- **max() -** Для нахождения «максимального» значения.

```
x = [99,24,55,67,89,100]
```

```
print(max(x))
```

```
100
```

- **min() -** Для нахождения «минимального» значения.

```
print(min(x))
```

```
24
```

Примеры Функций

- **len()** - Эта функция используется для вычисления длины последовательности или итерируемого объекта.

```
x = (8, 2, 12, 6, 7)
print(len(x))
```

```
5
```

```
len("Hello")
```

```
5
```

Примеры Функций

- **ist() -** Принимает итерируемый объект и возвращает список. Она обеспечивает большие гибкость и скорость при создании списков по сравнению с обычным способом.

```
list("Привет")
```

```
['П', 'р', 'и', 'в', 'е', 'т']
```

Примеры Функций

- **map() -** Используется для применения определенной функции к итерируемому объекту. Она возвращает результат в виде итерируемого объекта (списки, кортежи, множества). Можно передать и несколько объектов, но в таком случае нужно будет и соответствующее количество функций

```
a = '1234567890'
print(sum(map(int,a)))
```

```
45
```

```
a,b = map(int,input().split())
```

```
10 20
```

```
print(a,b)
```

```
10 20
```

Примеры Функций

- **next() -** Используется для итерируемых объектов. Умеет получать следующий (next) элемент в последовательности. Добравшись до конца, выводит значение по умолчанию.

```
lis = ['a', 'b', 'c', 'd', 'e']
```

```
x = iter(lis)
```

```
next(x)
```

```
'a'
```

```
next(x)
```

```
'b'
```

Примеры Функций

- **ord() -** Принимает один символ или строку длиной в один символ и возвращает соответствующее значение Unicode.

```
ord('F')
```

```
70
```

```
ord('ы')
```

```
1067
```

Примеры Функций

- **reversed() -** Эта функция предоставляет простой и быстрый способ развернуть порядок элементов в последовательности. В качестве параметра она принимает валидную последовательность, например список, а возвращает итерируемый объект.

```
x = [10,3,5,11]
```

```
b = reversed(x)
```

```
list(b)
```

```
[11, 5, 3, 10]
```

Примеры Функций

- **range() -** Используется для создания последовательности чисел с заданными значениями от и до, а также интервалом.

```
list(range(30,50,2))
```

```
[30, 32, 34, 36, 38, 40, 42, 44, 46, 48]
```

- **sorted() -** Используется для сортировки последовательностей значений разных типов. Например, может отсортировать список строк в алфавитном порядке или список числовых значений по возрастанию или убыванию.

```
x = [4, 5, 7, 3,42, 44, 46, 48]
sorted(x)
```

```
[3, 4, 5, 7, 42, 44, 46, 48]
```

Примеры Функций

- **str() -** Используется для создания строковых представлений объектов, но не меняет сам объект, а возвращает новый. У нее есть встроенные механизмы кодировки и обработки ошибок, которые помогают при конвертации.

```
str(10)
```

```
'10'
```

```
X = [1,4,5]
str(X)
```

```
'[1, 4, 5]'
```

Примеры Функций

- **set() -** Используется для создания наборов данных, которые передаются в качестве параметра. Обычно это последовательность, например строка (или список), которая затем преобразуется в множество уникальных значений.

```
set("Hello")
```

```
{'H', 'e', 'l', 'o'}
```

Примеры Функций

- **sum() -** Вычисление суммы — стандартная задача во многих приложениях. И для этого в Python есть встроенная функция. Она автоматически суммирует все элементы и возвращает сумму.

```
x = [30, 32, 34, 36, 38, 40, 42, 44, 46, 48]
sum(x)
```

```
390
```

Примеры Функций

- **type() -** Применяется в двух сценариях. Если передать один параметр, то она вернет тип этого объекта.

```
type(10)
```

```
int
```

```
type([10])
```

```
list
```

Математические операции

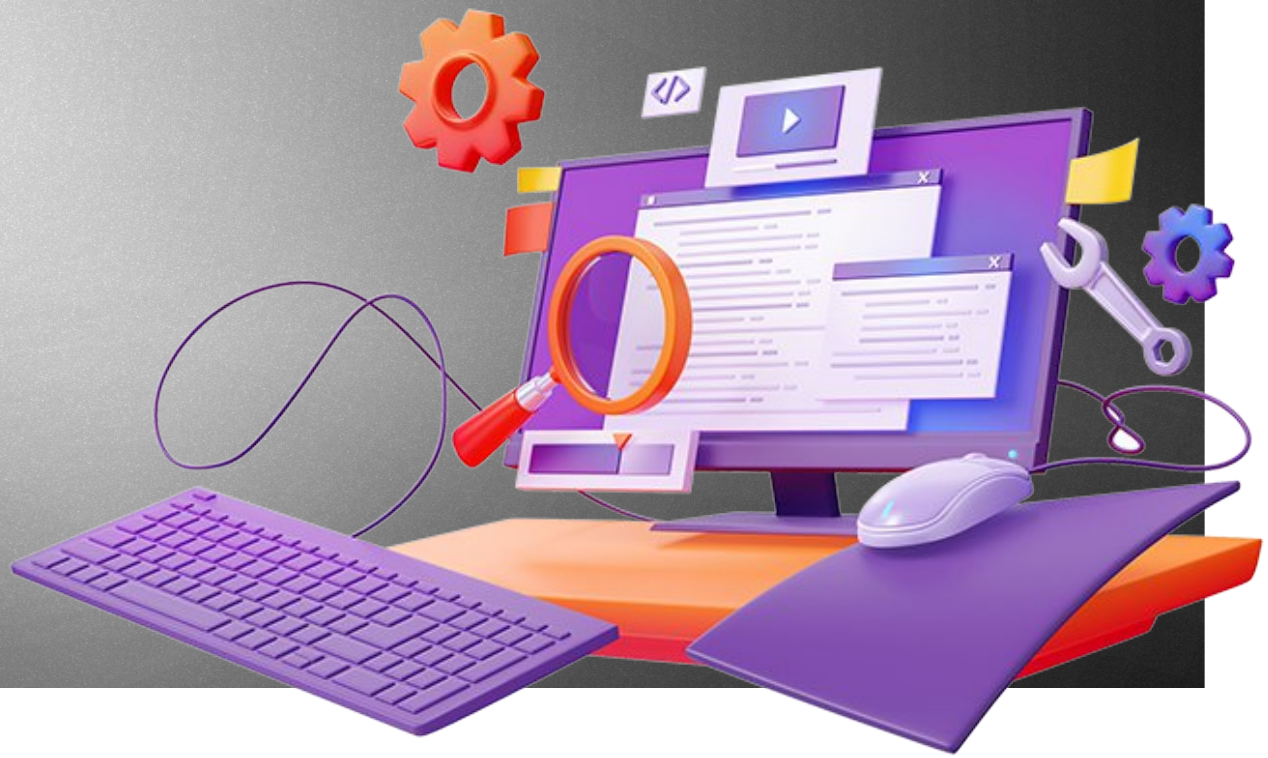
Описание	Оператор
сложение	+
вычитание	-
умножение	*
деление	/
возведение в степень	**
остаток от деления	%
целочисленное деление	//

```
import math
```

```
help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.
```

Спасибо за внимание.



Изображения (слайды 1, 38):
https://ru.freepik.com/