Горячие клавиши

Edit Mode (press Enter to enable)

- Ctrl-A : select all.
- Ctrl Home : go to cell start.
- Ctrl End : go to cell end.
- Ctrl Left : go one word left.
- Ctrl Right : go one word right.

- Tab : code completion or indent
- Shift-Tab : tooltip

In [23]:
```
pow()
```

Out[23]:
```
8
```

In [1]:
```
?pow
```

- Ctrl-Z : undo
- Ctrl-/ : comment
- Ctrl-D : delete whole line

In [ ]:
```
value = 3
value += 1
```

Command Mode (press Esc to enable)

- Up : select cell above
- Down : select cell below

- Shift-Enter : run cell, select below
- Ctrl-Enter : run selected cells
- Alt-Enter : run cell and insert below

- A : insert cell above
- B : insert cell below
- X : cut selected cells
- C : copy selected cells
- V : paste cells below

- F : find and replace

In [4]:
```
value = 5
value += 1
```

```
value -= 10
```

- Y : change cell to code
- M : change cell to markdown

- 1 : change cell to heading 1
- 2 : change cell to heading 2
- 3 : change cell to heading 3
- 4 : change cell to heading 4
- 5 : change cell to heading 5
- 6 : change cell to heading 6

- L : toggle line numbers
- H : show keyboard shortcuts
- I,I : interrupt the kernel
- 0,0 : restart the kernel (with dialog)

In [5]:
```
a = 12
3 / 0
b = 3
a + b
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-5-526983b54e29> in <module>
      1 a = 12
----> 2 3 / 0
      3 b = 3
      4 a + b

ZeroDivisionError: division by zero
```

In [21]:
```
for i in range(1000000000):
    pass
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-21-01acf624e1ef> in <module>
----> 1 for i in range(1000000000):
      2     pass

KeyboardInterrupt:
```

> **Markdown** - облегчённый язык разметки, созданный с целью обозначения форматирования в простом тексте, с максимальным сохранением его читаемости человеком, и пригодный для машинного преобразования в языки для продвинутых публикаций.

жирный

fdgdfg
курсив
fgfd

- список
- список

Latex

$$\frac{1}{n}\sum x_i j$$

> **Тэг HTML** - элемент языка разметки. Текст, содержащийся между начальным и конечным тегом, отображается и размещается в соответствии со свойствами, указанными в начальном теге.

 **жирный**

курсив

текст
а вот и новая строка

первый столбик	второй столбик
первый столбик	второй столбик

Словари

> **Словарь** - неупорядоченная структура данных, которая позволяет хранить пары «ключ — значение».

In [17]:
```
client_data = {
    'name': 'John',
    'surname': 'Doe',
    'age': 32
}

client_data
```

Out[17]:
```
{'name': 'John', 'surname': 'Doe', 'age': 32}
```

In [5]:
```
client_data['name']
```

Out[5]:
```
'John'
```

In [6]:
```
client_data.keys()
```

Out[6]:
```
dict_keys(['name', 'surname', 'age'])
```

In [8]:
```
client_data.values()
```

Out[8]:
```
dict_values(['John', 'Doe', 32])
```

In [7]:
```
client_data.items()
```

Out[7]:
```
dict_items([('name', 'John'), ('surname', 'Doe'), ('age', 32)])
```

In [11]:
```
for k, v in client_data.items():
    print(f'{k}: {v}')
```

```
name: John
surname: Doe
age: 32
```

In [38]:
```
client_data['height'] = 186
client_data
```

Out[38]:
```
{'gender': 'Male', 'age': 14, 'height': 186}
```

In [39]:
```
client_data['age'] = 14
client_data
```

Out[39]:
```
{'gender': 'Male', 'age': 14, 'height': 186}
```

In [1]:
```
client_data = {
    'name': 'John',
    'surname': 'Doe',
    'age': 32
}

client_data
```

Out[1]:
```
{'name': 'John', 'surname': 'Doe', 'age': 32}
```

In [2]:
```
client_data.setdefault('age', 14)
```

Out[2]:
```
32
```

In [3]:
```
client_data
```

Out[3]:
```
{'name': 'John', 'surname': 'Doe', 'age': 32}
```

In [4]:
```
client_data.update({
    'age': 14
})
client_data
```

Out[4]:
```
{'name': 'John', 'surname': 'Doe', 'age': 14}
```

> **Вложенный словарь** – это словарь, содержащий другие словари.

In [5]:
```
client_data.update({
    'age': 15,
    'body': {
        'weight': 80,
        'height': 186
    }
})
```

```
client_data
```

Out[5]:
```
{'name': 'John',
 'surname': 'Doe',
 'age': 15,
 'body': {'weight': 80, 'height': 186}}
```

In [6]:
```
client_data['body']
```

Out[6]:
```
{'weight': 80, 'height': 186}
```

In [7]:
```
client_data['body']['height']
```

Out[7]:
```
186
```

Модуль Random

Модуль random предоставляет функции для генерации случайных чисел или к примеру, для случайного выбора элементов последовательности. Посмотрим на самые полезные функции.

In [25]:
```
import random
```

In [38]:
```
random.randint(0, 10)
```

Out[38]:
```
9
```

In [40]:
```
random.seed(9)
random.randint(0, 10)
```

Out[40]:
```
7
```

In [11]:
```
random.random()
```

Out[11]:
```
0.9772194595909891
```

In [13]:
```
random.uniform(0, 10)
```

Out[13]:
```
7.835047467372638
```

In [41]:
```
a = ['black', 'red', 'yellow', 'green']
random.sample(a, 2)
```

Out[41]:
```
['yellow', 'red']
```

In [42]:
```
random.shuffle(a)
a
```

Out[42]:
```
['green', 'yellow', 'black', 'red']
```

In [43]:
```
random.choice(a)
```

Out[43]:
```
'black'
```

Функции

> **Функция в python** - объект, который принимает аргументы, производит с ними желаемые действия и возвращает значение. Также функция — это фрагмент программного кода, который решает какую-либо задачу. Его можно вызывать в любом месте основной программы. Функции помогают избегать дублирования кода при многократном его использовании.

> **Аргумент функции** — значение, которое передается в функцию при её вызове.

```
def func(arg1, arg2):
    # make magic
    result = arg1 + arg2
    return result
```

Аргументы по позиции и имени

> **Позиционный аргумент** - это аргумент, передаваемый в функцию в определенной последовательности (на определенных позициях), без указания их имен

In [63]:
```
def calc_sum(a, b):
    print(f'a is {a}')
    print(f'b is {b}')
    result = a + b
    return result
```

In [50]:
```
calc_sum(3, 5)
```

```
a is 3
b is 5
8
```
Out[50]:

> **Именованный аргумент** - это аргумент, передаваемые в функцию при помощи имени.

In [51]:
```
calc_sum(a=3, b=5)
```

```
a is 3
b is 5
8
```
Out[51]:

In [52]:
```
calc_sum(b=5, a=3)
```

```
a is 3
b is 5
8
```
Out[52]:

In [53]:
```
calc_sum(b=5, 3)
```

```
  File "<ipython-input-53-93aa6c5cb30e>", line 1
    calc_sum(b=5, 3)
                  ^
SyntaxError: positional argument follows keyword argument
```

In [54]:
```
calc_sum(3, b=5)
```

Out[54]:
```
a is 3
b is 5
8
```

Аргументы по умолчанию

В функцию можно передавать аргументы по умолчанию, это те аргументы, значения которых будут использованы, если не передали им явное значение при вызове.

In [55]:
```
def say_hi(greeting='Hello', name='World'):
    return f'{greeting}, {name}'
```

In [56]:
```
say_hi()
```

Out[56]:
```
'Hello, World'
```

In [58]:
```
def say_hi(greeting, name='World'):
    return f'{greeting}, {name}'
```

In [60]:
```
say_hi()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-60-6acd93c5568f> in <module>
----> 1 say_hi()

TypeError: say_hi() missing 1 required positional argument: 'greeting'
```

In [61]:
```
def say_hi(greeting='Hello', name='World'):
    return f'{greeting}, {name}'
```

In [62]:
```
say_hi(greeting='Hey', name='Apple')
```

Out[62]:
```
'Hey, Apple'
```

Аргумент *args

**args — это сокращение от «arguments» (аргументы).*

In [65]:
```
def calc_sum(a, b, c):
    result = a + b + c
    return result
```

In [66]:
```
def calc_sum(a, b, c, d, e):
    result = a + b + c + d + e
    return result
```

In [67]:
```
def calc_sum(*args):
    result = sum(args)
    return result
```

In [68]:
```
calc_sum(1)
```

Out[68]:
```
1
```

In [69]:
```
calc_sum(10, 0, 5)
```

Out[69]:
```
15
```

In [70]:
```
def calc_sum(*args):
    print(args)
    print(type(args))
    result = sum(args)
    return result
```

In [71]:
```
calc_sum(1, 2, 3)
```

Out[71]:
```
(1, 2, 3)
<class 'tuple'>
6
```

In [74]:
```
def printer(*args):
    print(type(args))
    print(args)
    print()
    for i in args:
        print(i)
```

In [75]:
```
printer(1, 'text', {'key': 'value'})
```

```
<class 'tuple'>
(1, 'text', {'key': 'value'})

1
text
{'key': 'value'}
```

In [76]:
```
names = ['John', 'Nick', 'Bill']
printer(names)
```

```
<class 'tuple'>
(['John', 'Nick', 'Bill'],)

['John', 'Nick', 'Bill']
```

In [77]:
```
printer(*names)
```

```
<class 'tuple'>
('John', 'Nick', 'Bill')

John
Nick
Bill
```

Атрибут **kwargs

***kwargs — сокращение от «keyword arguments» (именованные аргументы).*

In [81]:
```
def printer(**kwargs):
    print(type(kwargs))
    print()
```

```
    for key, value in kwargs.items():
        print(f'{key} - {value}')
```

In [82]:
```
printer(a=10, b='20')
```

```
<class 'dict'>

a - 10
b - 20
```

In [83]:
```
def say_hi(**kwargs):
    greeting = kwargs.get('greeting', 'Hello')
    name = kwargs.get('name', 'World')
    return f'{greeting}, {name}'
```

In [85]:
```
say_hi()
```

Out[85]:
```
'Hello, World'
```

In [87]:
```
say_hi(greeting='Hey', name='John')
```

Out[87]:
```
'Hey, John'
```

In [88]:
```
say_hi(greeting='Hey', name='John', extra=5)
```

Out[88]:
```
'Hey, John'
```

In [90]:
```
data = {'greeting': 'Hey',
        'name': 'John',
        'extra':5}

say_hi(**data)
```

Out[90]:
```
'Hey, John'
```

Аннотирование типов

In [96]:
```
def calc_sum(a: int, b: int) -> int:
    return a + b
```

In [97]:
```
calc_sum(1, 2)
```

Out[97]:
```
3
```

In [98]:
```
calc_sum(1.1, 2.2)
```

Out[98]:
```
3.3000000000000003
```

In [99]:
```
calc_sum('abc', 'dce')
```

Out[99]:
```
'abcdce'
```

Генераторы

Допустим, у вас есть файл, который весит десяток гигабайт. Из него нужно выбрать и обработать строки, подходящие под какое-то условие. Что в такой ситуации делать? А можно обрабатывать такие объемы данных небольшими порциями, чтобы не вызывать

переполнения памяти. В Python на этот случай есть специальный инструмент — генераторы.

> **Генератор** - это объект, который сразу при создании не вычисляет значения всех своих элементов.

Генератор хранит в памяти:

1. последний вычисленный элемент
2. правило перехода к следующему
3. условие, при котором выполнение прерывается

Вычисление следующего значения происходит лишь при выполнении метода next(). Предыдущее значение при этом теряется. Этим генераторы отличаются от списков, ведь списки хранят в памяти все свои элементы, и удалить их можно только программно. Вычисления с помощью генераторов экономят память.

In [6]:
```
def my_range(start, end):
    current = start
    while current < end:
        yield current
        current += 1
```

Оператор **yield** приостанавливает функцию и сохраняет локальное состояние, чтобы его можно было возобновить с того места, где оно было остановлено.

In [7]:
```
ranger = my_range(0, 2)
ranger
```

Out[7]:
```
<generator object my_range at 0x7f137c05c580>
```

In [8]:
```
next(ranger)
```

Out[8]:
```
0
```

In [9]:
```
next(ranger)
```

Out[9]:
```
1
```

In [10]:
```
next(ranger)
```

```
---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)
<ipython-input-10-c186e1cb4ee7> in <module>
----> 1 next(ranger)

StopIteration:
```

In [13]:
```
gen_a = my_range(2, 5)
for i in gen_a:
    print(i)
```

```
2
3
4
```

In [14]:
```
for i in gen_a:
    print(i)
```

list comprehensions

In [20]:
```
squares = []
for i in range(5):
    squares.append(i * i)
squares
```

Out[20]:
```
[0, 1, 4, 9, 16]
```

In [21]:
```
squares = [i * i for i in range(5)]
squares
```

Out[21]:
```
[0, 1, 4, 9, 16]
```

In [ ]:
```
# new_list = [expression for member in iterable]
```

In [23]:
```
even_nums = []
for i in range(10):
    if i % 2 == 0:
        even_nums.append(i)

even_nums
```

Out[23]:
```
[0, 2, 4, 6, 8]
```

In [25]:
```
even_nums = [i for i in range(10) if i % 2 == 0]
even_nums
```

Out[25]:
```
[0, 2, 4, 6, 8]
```

In [26]:
```
orig_prices = [125, -9, 103, 38, -5, 116]
prices = [i if i > 0 else 0 for i in orig_prices]
prices
```

Out[26]:
```
[125, 0, 103, 38, 0, 116]
```

In [27]:
```
def get_price(price):
    return price if price > 0 else 0


prices = [get_price(i) for i in orig_prices]
prices
```

Out[27]:
```
[125, 0, 103, 38, 0, 116]
```

set comprehensions

In [28]:
```
numbers = ['black', 'red', 'green', 'green', 'red', 'green']
{i for i in numbers}
```

Out[28]:
```
{'black', 'green', 'red'}
```

dict comprehensions

In [29]:
```
squares_dict = {i: i * i for i in range(5)}
squares_dict
```

Out[29]:
```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

In [36]:
```
%%time

even_nums = []
for i in range(100_000_000):
    if i % 2 == 0:
        even_nums.append(i)
```

```
CPU times: user 10.6 s, sys: 373 ms, total: 11 s
Wall time: 11 s
```

In [37]:
```
%%time
even_nums = [i for i in range(100_000_000) if i % 2 == 0]
```

```
CPU times: user 3.74 s, sys: 441 ms, total: 4.18 s
Wall time: 4.18 s
```