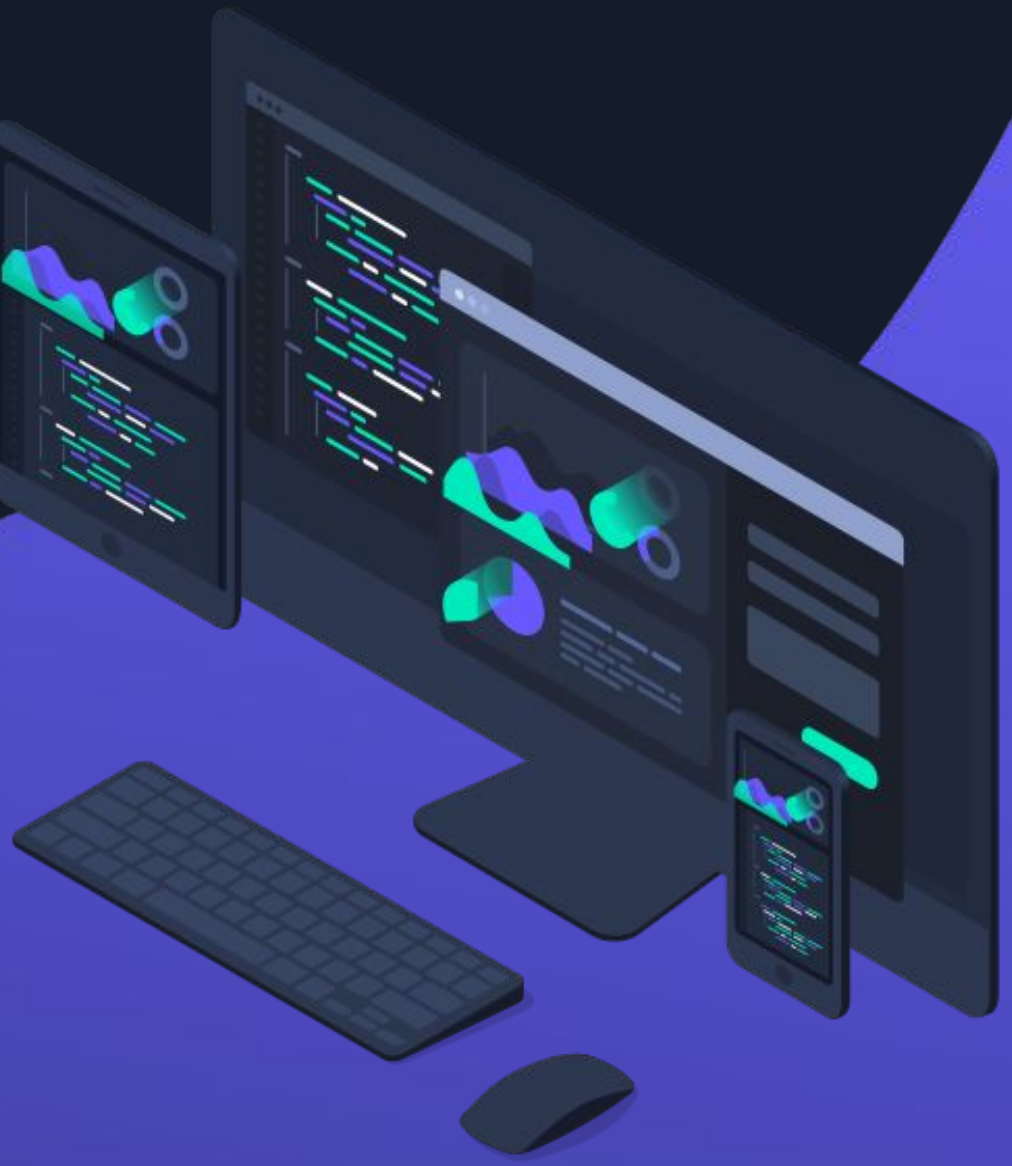
ITEA ONLINE



ITEA ONLINE

Николай Свиридов

- **Python Backend Developer**

КОНТАКТНЫЕ ДАННЫЕ

https://www.linkedin.com/in/luchanos/

t.me/luchanos

Vk.com/luchanos

Типы данных

Тип	Пример
int	2, 4, 8, -10, -2
float	2.6, -5.2
str	“my_text”
bool	True, False
list	[2, 2.4, “Hello”]
tuple	(2, 2.4, “Hello”)
dict	{“name”: “Вася”, “age”: 10}

Встроенные контейнеры

Список

- изменяемая упорядоченная коллекция с элементами произвольного типа.

```
l0 = list() or []  # пустой список
l1 = [1, 2, 3, "test string", ValueError]  # список с произвольным наполнением
```

Кортеж

- НЕизменяемая упорядоченная коллекция с элементами произвольного типа.

```
t0 = tuple() or ()  # пустой кортеж
t1 = (1, 2, 3, "test string", ValueError)  # кортеж с произвольным наполнением
```

Множество

- изменяемая неупорядоченная коллекция, содержащая уникальные элементы произвольного неизменяемого типа.

```
s0 = set()  # пустое множество
s1 = {1, 2, 3, "test string", ValueError}  # множество с произвольным наполнением
```

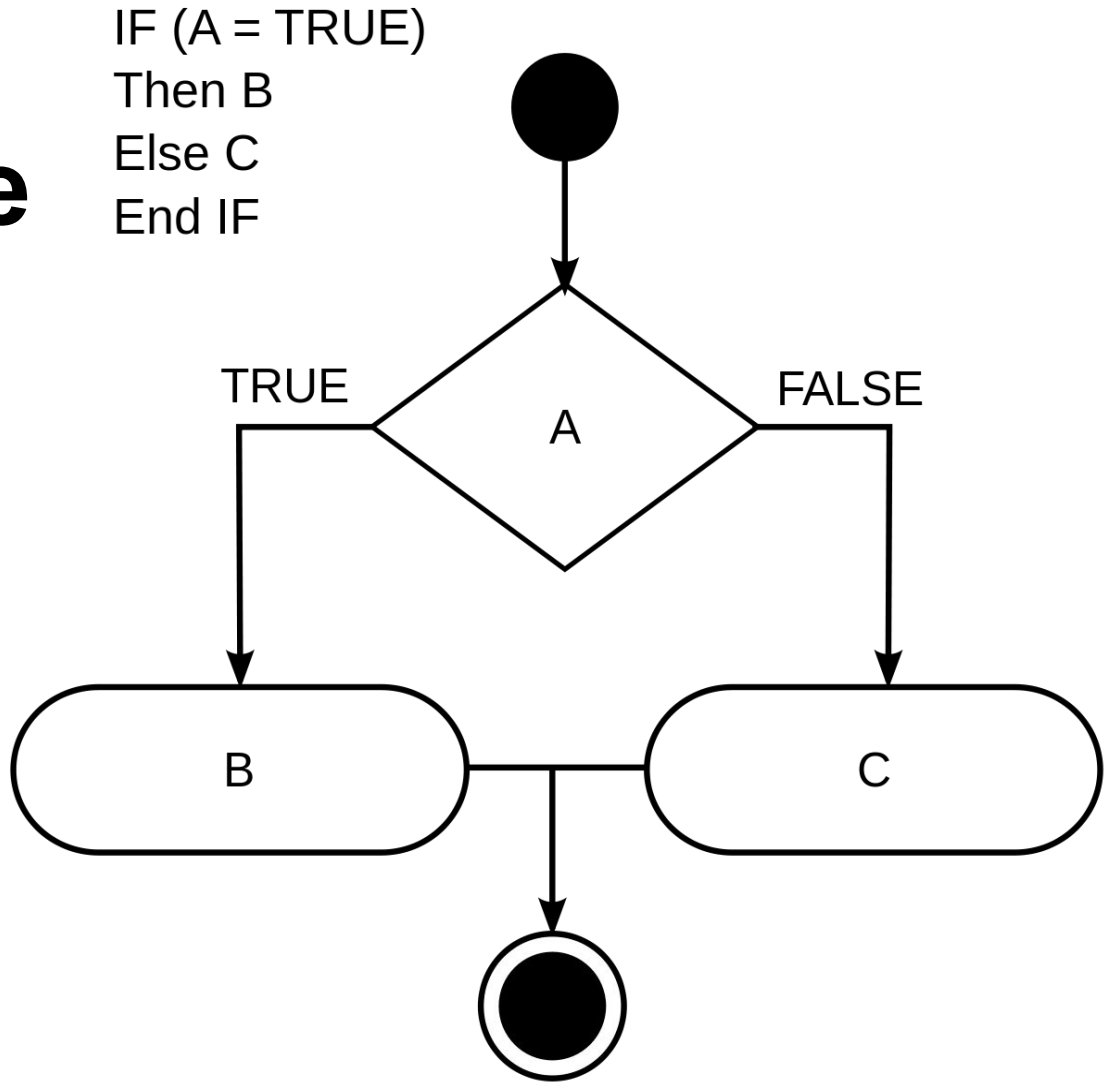
Словарь

- изменяемая неупорядоченная коллекция типа "ключ" - "значение", где ключ - произвольный объект неизменяемого типа, а значение может быть любым объектом.

```
d0 = dict() or {}  # пустой словарь
d1 = {1: "One", 2: "Two", 3: "Three"}  # словарь с произвольным наполнением
```

Операторы контроля выполнения: if, elif, else

- механизм управления ходом исполнения программы в зависимости от условий - организуют ветвление вложенных сценариев.



Обработка ошибок

- управляем внештатными ситуациями и переводим их в разряд штатных

```
# обработка ошибок
f = open('1.txt', 'w')
res = []
try:
    line = int(input("Введите число: "))
    s = "Число %d" % line
    f.write(s)
except ValueError:
    print('Это не число. Выходим.')
except Exception:
    print('Это что ещё такое?')
else:
    print('Всё хорошо.')
finally:
    f.close()
    print('Я закрыл файл.')
# Именно в таком порядке:
# try, группа except,
# затем else, и только потом finally.
```

Операторы циклов. While.

- позволяет повторять блок кода до тех пор, пока уловие выполняется

```
while condition:  # какое-то условие
    print("Это")
    print("Блок")
    print("Кода")
    print("В цикле")
    if break_condition:
        print("Выходим из цикла")
        break
    if continue_condition:
        print("Сразу идем на новую итерацию")
    print("И кое-что в конце!")
else:
    print("Если вышли из цикла штатно")
```

Операторы циклов. For.

- позволяет поэлементно итерировать то, что итерируется (например коллекции)

```
l0 = ["Это", "Итерация", "Элементов", "Списка", "Хаха"]
for el in l0:
    print(el)
    if el == "Списка":
        break
    if el == "Итерация":
        continue
    print("И кое-что в конце!")
else:
    print("Если вышли штатно")
```

Comprehensions

- невероятная возможность генерировать коллекции "на лету"
- ещё их называют "генераторами списков/словарей" и т.д., но лучше их так не называть (а то путаница с генераторами-итераторами)
- работают быстрее, нежели формирование в простом цикле

Comprehensions

```
ls = [x for x in range(10)]
s = {x for x in range(10)}
g = (x for x in range(10))  # это не кортеж! это объект-генератор!
t = tuple(x for x in range(10))  # а вот это кортеж!
# а ещё можно делать словари:
d = {k: v ** 2 for k, v in enumerate(range(10))}
```

Создание функций и процедур

- Функция - это объект, принимающий аргументы и возвращающий значение. Это способ упаковать блок кода в "ящик" и воспроизводить его логику по вызову этого "ящика". Функции обеспечивают возможность использования блоков кода повторно в любой точке программы.

```
def bogatyr_choice(direction):
    if direction == "left":
        return "Horse loss!"
    elif direction == "right":
        return "Mind loss!"
    elif direction == "straight":
        return "Life loss!"
```

Аргументы функций

- Позиционные
- Именованные
- Обязательные
- Необязательные

```python
# вы точно знаете, сколько аргументов
# должна принимать ваша функция
# a и b - обязательные аргументы
def func_0(a, b):
    return a + b


# вы точно знаете, сколько аргументов
# должна принимать ваша функция
# a и b - НЕобязательные аргументы.
def func_1(a=0, b=0):
    return a + b
```

Аргументы функций (запускаем функцию)

При запуске функции можно передавать в неё аргументы позиционно или по имени (по ключу)

```
# передача аргументов по ключу
print(func_1(a=1, b=2))

# передача аргументов по позиции
print(func_1(1, 2))
```

Аргументы функций *args **kwargs

Используются в том случае, если предполагается что в ходе использования функции она будет принимать непостоянный по своему составу набор аргументов как по именам и позициям, так и по составу

```
# вы не знаете, сколько аргументов
# точно будет передано в вашу функцию
def func_2(*args, **kwargs):
    print("Доп. аргументы по позиции: ", args)
    print("Доп. аргументы по ключу: ", kwargs)
```

Домашнее задание

1. Написать декоратор, который будет печатать на экран время работы функции.
2. Написать функцию для вычислений очередного числа Фибоначчи (можно через цикл, можно через рекурсию).
3. Реализовать функцию, которая принимает три позиционных аргумента и возвращает сумму наибольших двух из них.
4. Написать программу, которая запрашивает у пользователя строку чисел, разделённых пробелом. При нажатии Enter должна выводиться сумма чисел. Пользователь может продолжить ввод чисел, разделённых пробелом и снова нажать Enter. Сумма вновь введённых чисел будет добавляться к уже подсчитанной сумме.
Но если вместо числа вводится специальный символ, выполнение программы завершается. Если специальный символ введён после нескольких чисел, то вначале нужно добавить сумму этих чисел к полученной ранее сумме и после этого завершить программу.

ITEA

ONLINE