Функции и декораторы python

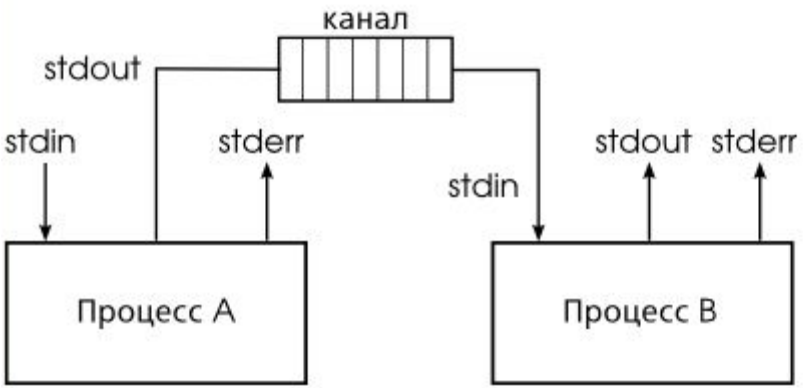
План вебинара:

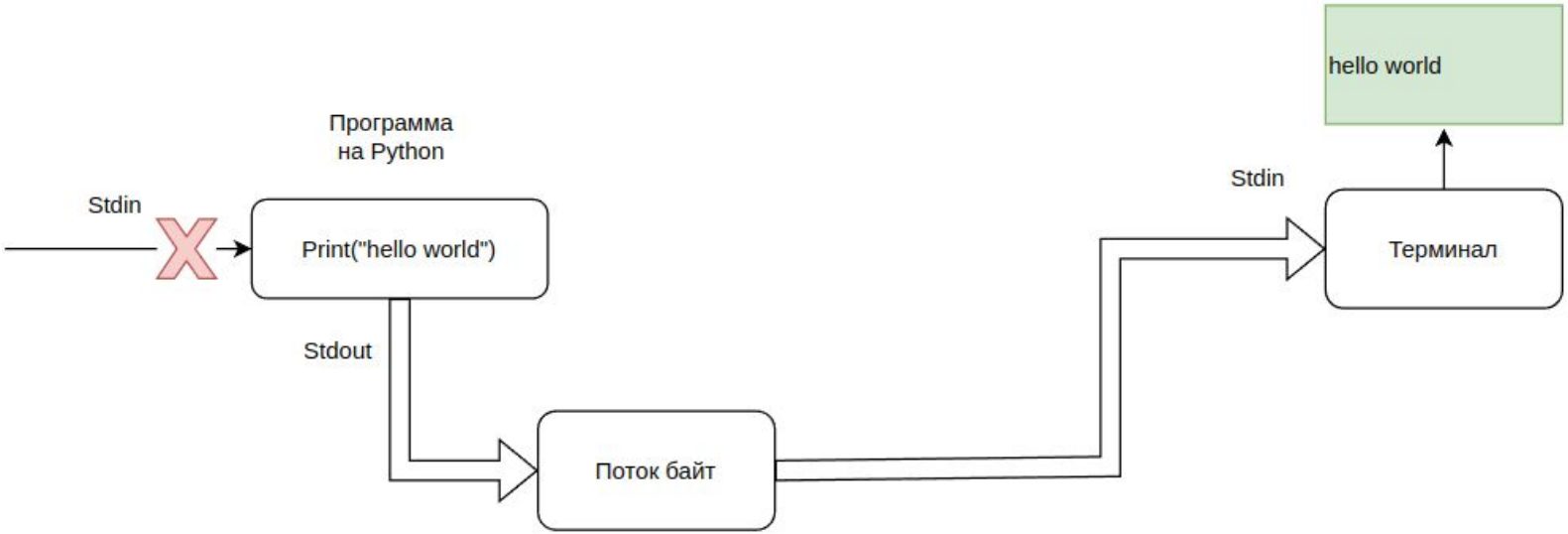
1. Ответы на вопросы
2. Функции и декораторы в python

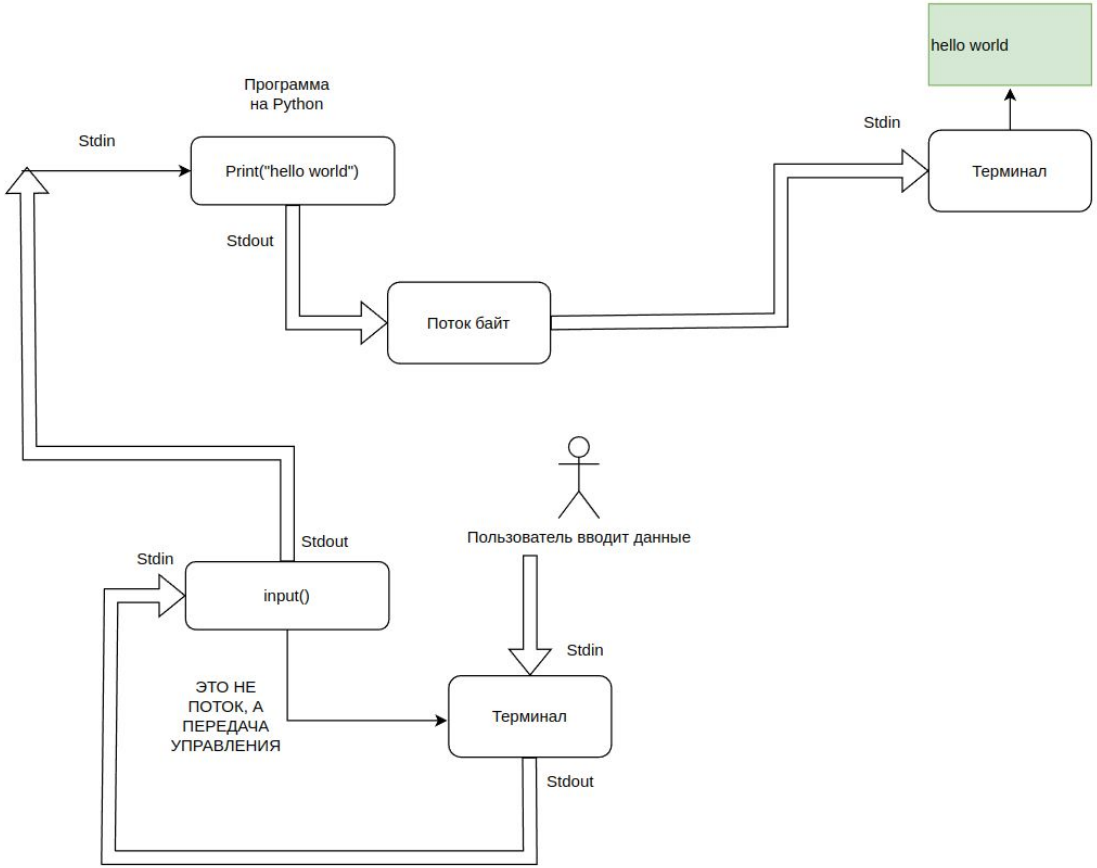
Ответы на вопросы

1. Как же все-таки работает этот print?
2. Про хоткеи и быструю печать
3. Примеры из модулей
4. Непонятные термины

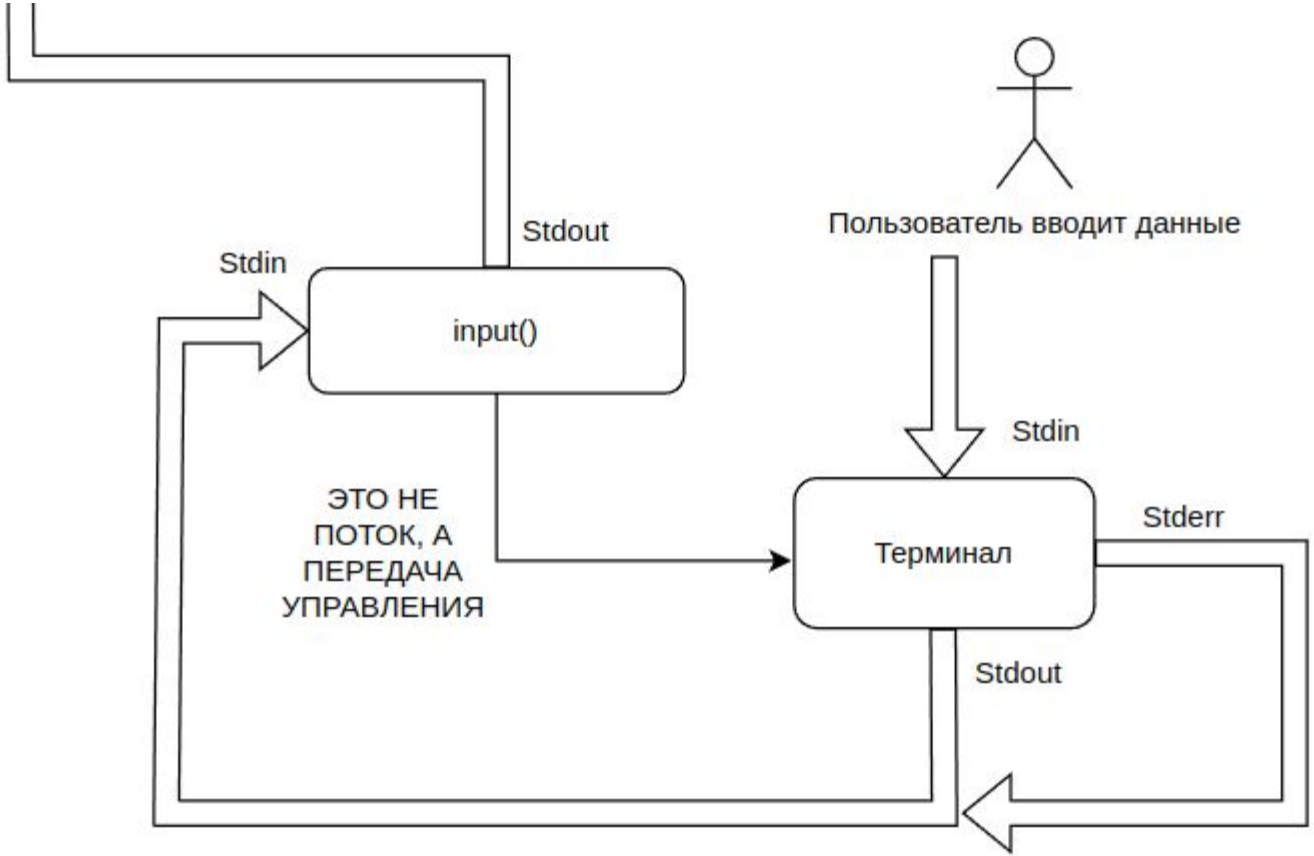
1. Print и потоки




hello world
Программа
на Python
Stdin
Print("hello world")
Stdout
Поток байт
Stdin
Терминал


hello world
Программа
на Python
Stdin
Print("hello world")
Stdout
Поток байт
Stdin
Терминал
Пользователь вводит данные
Stdout
Stdin
input()
ЭТО НЕ
ПОТОК, А
ПЕРЕДАЧА
УПРАВЛЕНИЯ
Stdin
Терминал
Stdout

2>&1


Stdout
Stdin
input()
Пользователь вводит данные
Stdin
ЭТО НЕ
ПОТОК, А
ПЕРЕДАЧА
УПРАВЛЕНИЯ
Терминал
Stderr
Stdout

Доказательство:

Input.py:

```
import sys

for line in sys.stdin:

 print("Мой stdin поймал ваш stdout "+line)
```

Output.py:

```
print("Hello world")
```

python3 output.py | python3 input.py

Ч.Т.Д

Хоткеи, быстрая печать:

Ctrl+Ctrl+стрелочка - несколько курсоров

Help > Keyboard Shortcuts

https://monkeytype.com/

Примеры из модулей:

print(revers(абырвылг)) -так ругается. NameError: name 'абырвылг' is not defined

print(revers("абырвылг")) - так проходит почему?

```
def revers(text):
    if len(text) == 0:
        print("Пустое выражение")
        return text # Почему когда комментирую  здесь return, функция ломается.Зачем два вывода return в функции? вывод должен быть на каждое условие?
    else:
        return text[-1] + revers(text[:-1])
print(revers("абырвылг"))
```

4. Не понятно как функция в функции работает. Принцип вроде понятен, но в голове не получается удержать как обрабатывается переменная на каждой интерации. С числами вроде стыкуется.

```
def sum_digit(n):
    if n < 10:
        return n
    else:
        return n % 10 + sum_digit(n // 10)
print(sum_digit(211))
```

Почему тут не получается сделать произведение любоко кол-ва цифр поданных на вход функции? запутался с преобразованием и пониманием типа переменных

```
def proizv(*all_num):
    nums = 1
    # all_num = all_num.replace(" ", "")
    # all_num = all_num.replace(", ", "")
    # all_num = all_num.replace("\n", "")
    List_num = list(map(int, all_num))
    print(List_num)
    for n in List_num:
        nums *=n
    return nums
print("произведение чисел = ", proizv(2, 2, 2))
```

У лукоморья…

```
N = 5


for i in range(1, N + 1):
    print("*" * i)
```

Фибо

Предсказываем будущее с дебаггером и фибоначчи

```
def rec_fibb(n):
  if n == 1:
      return 1
  if n == 2:
     return 1
  return rec_fibb(n - 1) + rec_fibb(n - 2)


print(rec_fibb(5)) # 5
```

Термины,методы,команды

Словари, списки, кортежи

```
text = text.lower()

text = text.replace(" ", "")

text = text.replace("\n", "")
```

Декораторы в python

```
def new_decorator(function_to_decorate):
    # Внутри себя декоратор определяет функцию-"обёртку". Она будет обёрнута вокруг декорируемой,
    # получая возможность исполнять произвольный код до и после неё.
    def the_wrapper_around_the_original_function():
        print("Отработает до вызова функции")
        function_to_decorate() # Сама функция (stand_alone_function)
        print("Срабатывает после")
    # Вернём обертку
    return the_wrapper_around_the_original_function

 # Представим теперь, что у нас есть функция, которую мы не будем изменять.
def stand_alone_function():


    print("Функция, работающая сама по себе")
```

```
def make_Cocktail(func):
    def wrapper(*args):
        print("Ингредиенты для коктейля:")
        print("Мороженое, Молоко, Бананы")
        print("Рецепт:\n"
              "1. Залить молоко в миксер\n"
              "2. Добавить мороженое в молоко\n"
              "3. Нарезать бананы\n"
              "4. Запустить миксер\n")
        func(args, 'Молоко', 'Бананы', 'Мороженое')

    return wrapper


def make_Meat(func):
    def wrapper(*args):
        print("Ингредиенты для мяса:")
        print("Мясо, Специи, Лук")
        print("Рецепт:\n"
              "1. Пожарить лук\n"
              "2. Пожарить мясо\n"
              "3. Добавить специи\n")
        func(args, 'Мясо', 'Специи', 'Лук')

    return wrapper


@make_Cocktail
@make_Meat
def cook_Dinner(*args):
    print(f"Итоговый список покупок: {args} ")


cook_Dinner()
```