Практика 1. Исполнение и окружения

материалы преподавателя

Управляющие структуры

Управляющие структуры описывают логическую последовательность исполнения программы. Например, инструкция `if-elif-else` позволяет пропускать некоторую часть кода, инструкция `while` — многократно повторять один и тот же фрагмент.

Инструкция `if`

Условные инструкции позволяют исполнять в программе различные наборы инструкций в зависимости от некоторых условий. Рассмотри подробнее синтаксис инструкции `if-elif-else`:

```
if <условное выражение>:
    <набор инструкций>
elif <условное выражение>:
    <набор инструкций>
else:
    <набор инструкций>
```

Основные моменты:

- Предложения `else` и `elif` опциональны. Предложений `elif` может быть сколь угодно много.
- **Условное выражение** — это обычное выражение, но его результат будет рассматриваться как истинный (`True`, целое-число-не-ноль и так далее) или как ложный (`False`, `0`, `None`, `''`, `[]` и другие).
- **Набор инструкций** — часть кода с дополнительным отступом относительно `if/elif`, которая будет исполнена, если соответствующее **условное выражение** истинно.
- Если все условия не выполнены, то выполнится набор инструкций после `else`.

Булевы операторы

Python поддерживает три булевых оператора: `and`, `or` и `not`.

- `not True` имеет значение `False`, `not False` имеет значение `True`.
- `True and True` имеет значение `True`, но значение `False` с любой стороны приведет к тому, что всё выражение будет `False`.
- `False or False` имеет значение `False`, но если хотя бы с одной стороны будет `True` — всё выражение будет `True`.

```
>>> not None
True
>>> not True
False
>>> -1 and 0 and 1
0
>>> False or 9999 or 1/0
9999
```

Вопрос 1

Петя надевает куртку лишь тогда, когда температура на улице опускается ниже 15 градусов или когда идёт дождь.

Напиши функцию, которая принимает текущую температуру, булево значение (флаг) отражающее наличие дождя и возвращает `True`, если Петя надел куртку и `False`, если нет.

При решении используй инструкцию `if`.

```
def wears_jacket(temp, raining):
    """
    >>> wears_jacket(32, False)
    False
    >>> wears_jacket(4, False)
    True
    >>> wears_jacket(37, True)
    True
    """
```

Ответ

```
    if temp < 15 or raining:
        return True
    else:
        return False

Нужно объяснить и такое решение:

    return temp < 15 or raining
```

Цикл `while`

Для повторения одних и тех же инструкций множество раз, можешь использовать итерацию. Один из способов сделать это в Python — инструкция `while`.

```
while <условное выражение>:
    <набор инструкций>
```

Пока `<условное выражение>` будет истинным, выполнение `<набора инструкций>` будет повторяться. Каждый раз при достижении конца цикла значение `<условного выражения>` будет вычисляться вновь.

Вопрос 2

Что произойдёт в результате выполнения этого кода?

```
def square(x):
    print("тут!")
    return x * x

def so_slow(num):
    x = num
    while x > 0:
        x=x+1
    return x / 0

square(so_slow(5))
```

Ответ

```
Получится бесконечный цикл, поскольку x всегда будет больше 0; выражение x/0 никогда
на будет исполнено. Таким образом и строка «тут!» никогда не будет напечатана —
значение операнда so slow(5) должно быть вычислено до вызова square(x).
```

Вопрос 3

Напиши функцию возвращающую `True`, если положительное целое `n` является простым. В противном случае функция долна возвращать `False`.

Выражение `x % y` означает взять остаток от деления `x` на `y`.

```
def is_prime(n):
    """
    >>> is_prime(10)
    False
    >>> is_prime(7)
    True
    """
```

Ответ

```
    if n == 1:
        return False
    k = 2
    while k < n:
        if n % k == 0:
            return False
        k += 1
    return True

Альтернативно k < n может быть заменено на k <= sqrt(n) — это надо проговорить с
группой.
```

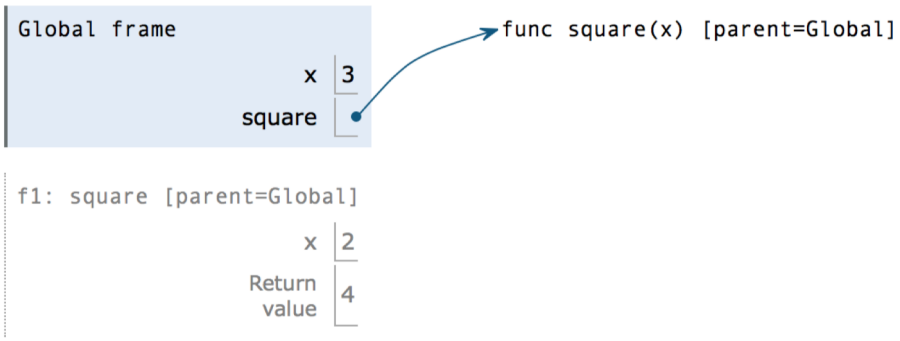
Диаграммы окружения

Диаграммы окружения позволяют увидеть все переменные и их значения, которые были заданы во время исполнения программы. Этот инструмент будет использоваться на протяжении всего курса для понимания сложных программ, включающих множество объектов и вызовов функций.

```
x=3

def square(x):
    return x ** 2

square(2)
```



Программа — это всего лишь набор инструкций и выражений. Диаграмма окружения, описывающая программу, состоит из тех же инструкций!

Инструкция присваивания

Инструкции присваивания, такие как `x = 3`, позволяют определять в программах переменные. Для выполнения такой инструкции на диаграмме окружения следует:

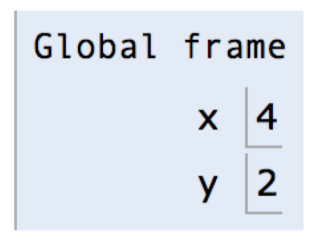
1. Вычислить значение выражения справа от знака `=`.
2. Записать имя переменной и значение выражения в текущий фрейм.

Вопрос 4

Используя вышеприведённые правила, нарисуй простую диаграмму для представленных ниже присвоений.

```
x = 10 % 4
y = x
x **= 2
```

Ответ



Инструкция `def`

Инструкция `def` создаёт функцию и связывает её с заданным именем. Для выполнения инструкции `def` на диаграмме нужно записать имя функции в текущий фрейм и связать его с функцией стрелочкой. Радом с функцией важно указывать **parent frame** (родительский фрейм) — имя того фрейма, в котором была создана функция.

1. Изобрази функцию справа от фреймов, то есть запиши имя функции, имена её формальных параметров и ссылку на родительский фрейм (то есть `func square(x) [parent = Global]`).
2. Запиши имя функции в текущий фрейм и нарисуй стрелочку от этого имени к функции.

Вопрос 5

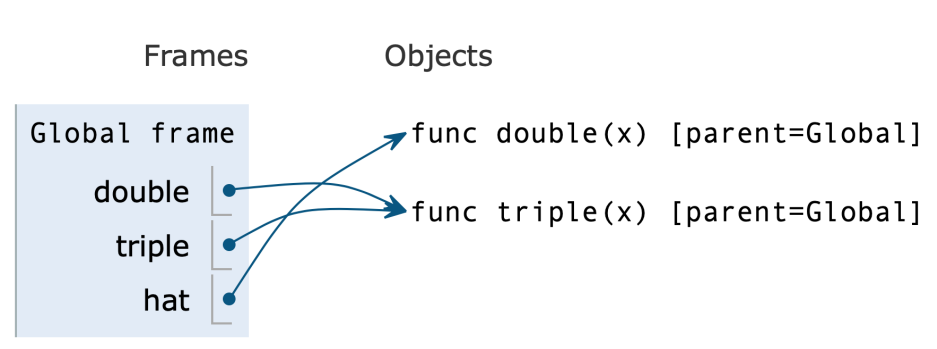
Используя вышеприведённые правила, нарисуй диаграмму для программы.

```
def double(x):
    return x * 2

def triple(x):
    return x * 3

hat = double
double = triple
```

Ответ



Следует обратить внимание группы на различия во внутреннем и внешнем имени функции.

Вызывающие выражения

Вызывающие выражения, такие как `square(2)`, вызывают выполнение функции с заданными аргументами. При выполнении вызывающего выражения на диаграмме окружения создаётся новый фрейм для учёта локальных переменных (тех, что определяются внутри функции):

1. Определяется значение оператора (значением должна оказаться некоторая функция).
2. Вычисляются значения аргументов (слева направо).
3. На диаграмме изображается новый фрейм, снабжённый:
   - Уникальным идентификатором (f1, f2, f3, ...)
   - **Внутренним** именем функции, которое является собственным именем функции. Например, у функции `func square(x) [parent=Global]` внутренним именем является `square`.
   - Указанием на родительский фрейм (то есть `[parent=Global]`)
4. В созданный фрейм записываются имена формальных параметров и связываются со значениями аргументов (пункт 2)
5. Тело функции выполняется в контексте созданного фрейма до тех пор, пока не встретится инструкция `return` (явно или неявно). Значение возвращаемого выражения записывается в текущий фрейм.

Если в теле функции отсутствует инструкция `return`, то результатом выполнения этой функции будет значение `None`.

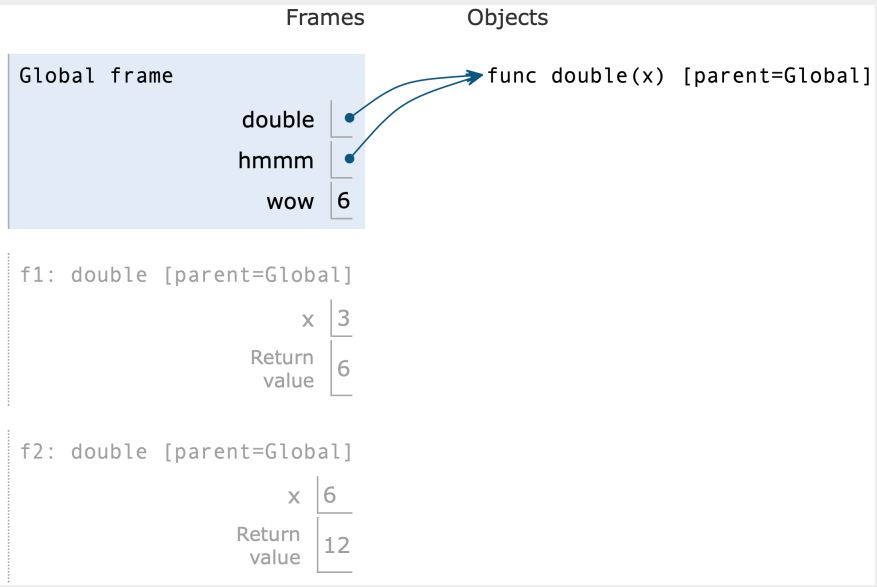
Вопрос 6

Теперь собери все эти факты вместе. Нарисуй диаграмму окружения для следующего кода.

```
def double(x):
    return x * 2

hmmm = double
wow = double(3)
hmmm(wow)
```

Ответ



Вопрос 7

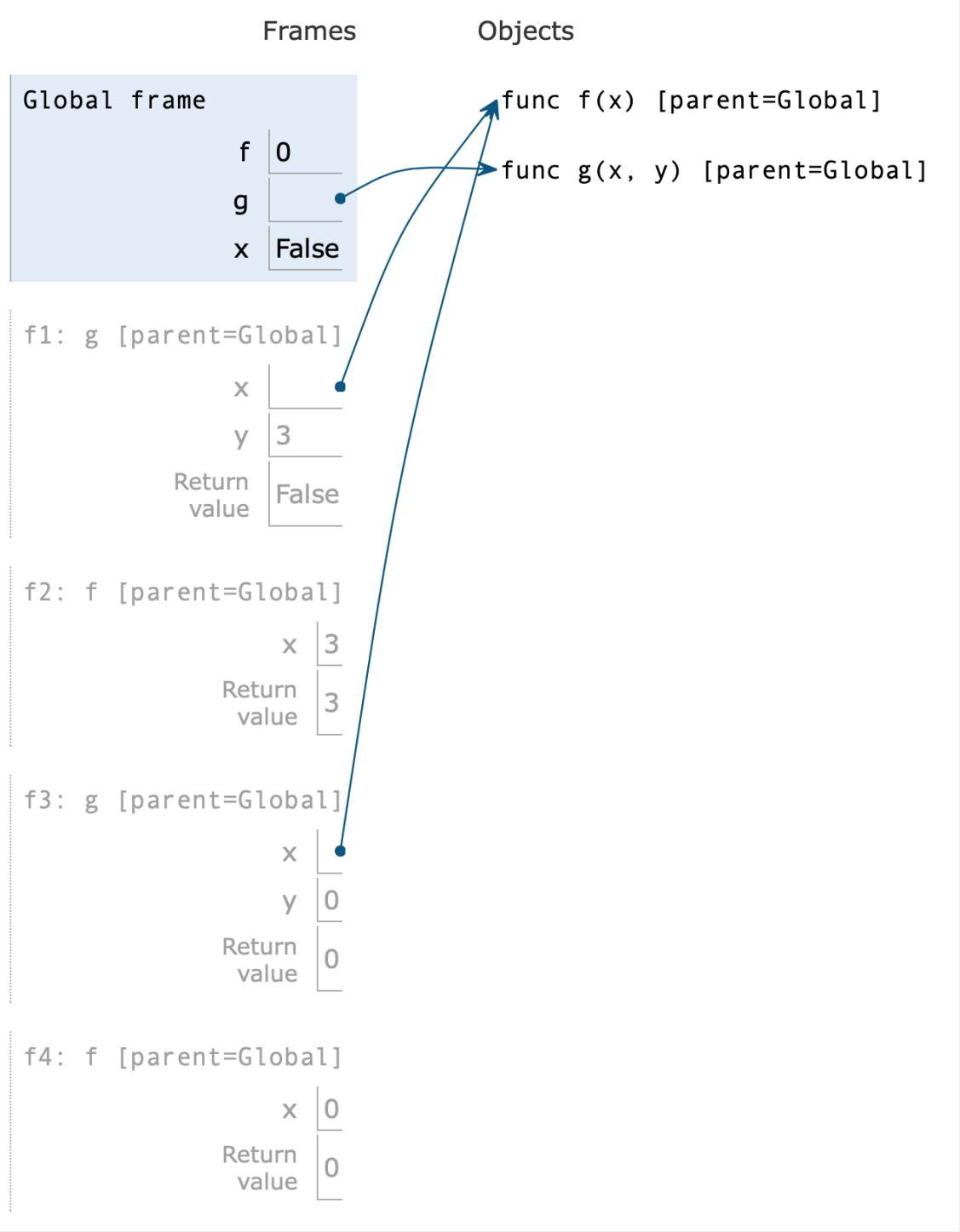
Нарисуй диаграмму окружения для нижеприведённого кода.

```
def f(x):
    return x

def g(x, y):
    if x(y):
        return not y
    return y

x = 3
x = g(f, x)
f = g(f, 0)
```

Ответ


Frames
Objects
Global frame
f
0
g
x
False
func f(x) [parent=Global]
func g(x, y) [parent=Global]
f1: g [parent=Global]
x
y
3
Return value
False
f2: f [parent=Global]
x
3
Return value
3
f3: g [parent=Global]
x
y
0
Return value
0
f4: f [parent=Global]
x
0
Return value
0