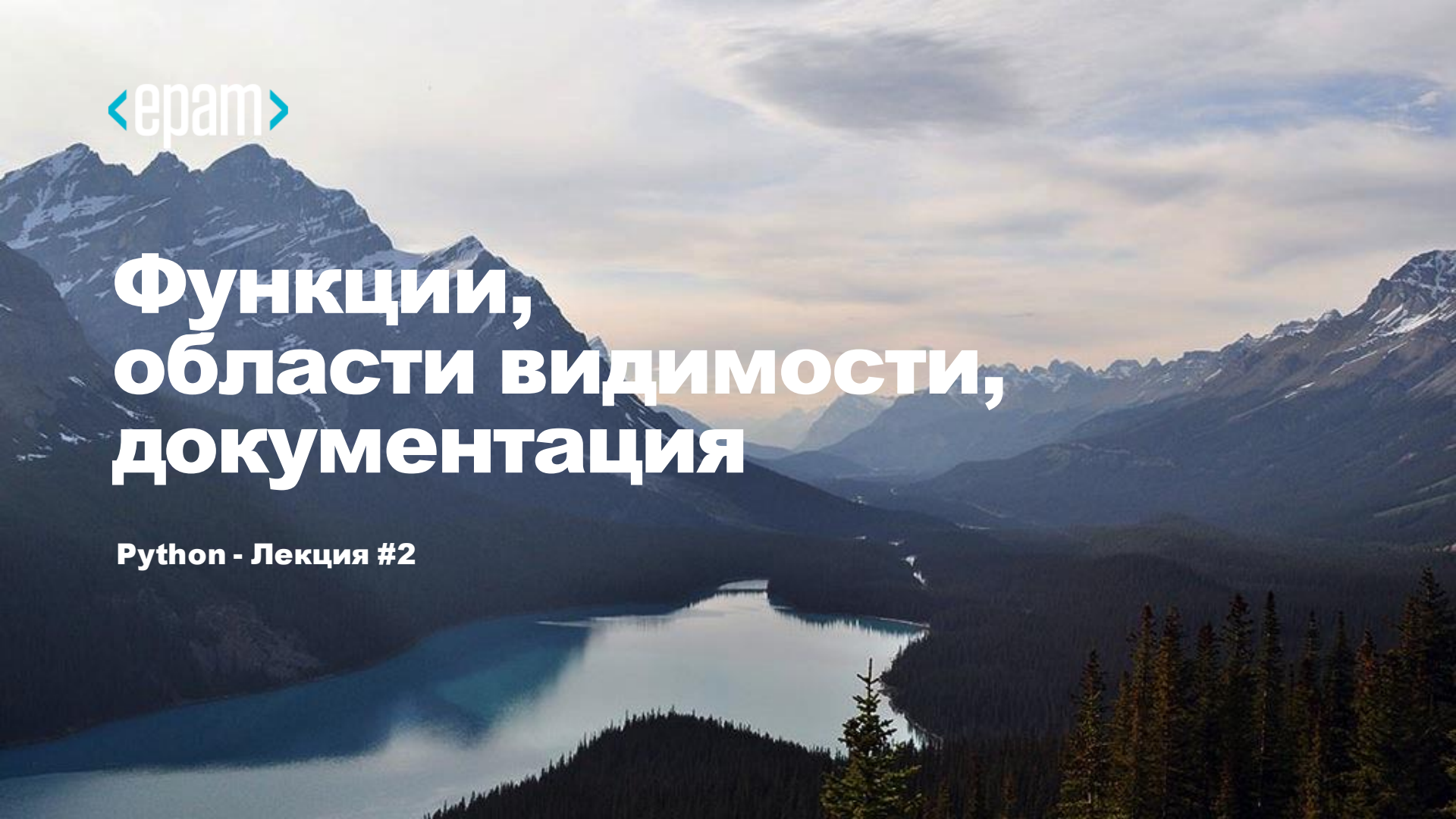


Функции, области видимости, документация

Python - Лекция #2

Синтаксис определения функций - 0

```
def <name>(arg1, arg2,... argN):
    <statements>
```

```
def <name>(arg1, arg2,... argN):
    ...
    return <value>
```

Синтаксис определения функций - 1

Ограничения на имя функций в Python типичны:

- можно использовать буквы
- подчеркивание _
- цифры от 0 до 9 (цифра не должна стоять на первом месте)

return использовать не обязательно, по умолчанию функция возвращает None

Функция - это объект

Инструкция def в языке Python – это настоящая исполняемая инструкция: когда она исполняется, она создает новый объект функции и присваивает этот объект имени

```
>>> def foo():
...     pass
...
>>> type(foo)
<type 'function'>
```

Передача аргументов

- Аргументы передаются через автоматическое присваивание объектов локальным переменным

- Изменение внутри функции аргумента, который является изменяемым объектом, может оказывать влияние на вызывающую программу (!)

Позиционные аргументы

```
>>> def min(a, b):
...     if a < b:
...         return a
...     else:
...         return b
>>> min(a=-5, b=12)
-5
```

Позиция аргументов важна!

Позиционные аргументы: произвольное число

```
>>> def min(*args): # type(args) == tuple.
...     res = float("inf")
...     for arg in args:
...         if arg < res:
...             res = arg
...     return res

>>> min(-5, 12, 13)
-5
>>> min(1, 2, 3)
inf
```

Как потребовать, чтобы в args был хотя бы один элемент?

Позиционные аргументы: произвольное число

```
>>> def min(first, *args):
...     res = first
...     # ...
...
>>> min()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: min() missing 1 required [...] argument: 'first'
```

Именованные аргументы

```
>>> def bounded_min(first, *args, lo=None, hi=None):
...     res = hi
...     for arg in (first, ) + args:
...         if arg < res and lo < arg < hi:
...             res = arg
...     return max(res, lo)
...
>>> bounded_min(-5, 12, 13, lo=0, hi=255)
12
```

Позиция аргументов не важна, но именованные аргументы ДОЛЖНЫ указываться после позиционных

Именованные аргументы: произвольное число

Ключевые аргументы, аналогично позиционным, можно упаковывать и распаковывать:

```
>>> def runner(cmd, **kwargs):
...     if kwargs.get("verbose", True):
...         print("Logging enabled")
...
>>> runner("mysqld", limit=42)
Logging enabled
>>> runner("mysqld", **{"verbose": False})
>>> options = {"verbose": False}
>>> runner("mysqld", **options)
```

Именованные аргументы: значения по умолчанию

```
>>> def bounded_min(first, *args, lo=float("-inf"),
...                 hi=float("inf")):
...     res = hi
...     for arg in (first, ) + args:
...         if arg < res and lo < arg < hi:
...             res = arg
...     return max(res, lo)
...
>>> bounded_min(-5, 12, 13, lo=0, hi=255)
12
```

Именованные аргументы: подводные камни

```
>>>def unique(iterable, seen=set()):
...     acc = []
...     for item in iterable:
...         if item not in seen:
...             seen.add(item)
...             acc.append(item)
...     return acc
...

>>>xs = [1, 1, 2, 3]
>>>unique(xs)
[1, 2, 3]

>>>unique(xs)
[]

>>>unique.__defaults__
({1, 2, 3},)
```

Именованные аргументы: подводные камни

```
>>> default = 2
>>> def foo(val = default):
...     print val
...
>>> foo(4)
4
>>> foo()
2
>>> default = 12
>>> foo()
2
```

Аннотации функций

```
>>> def greeting(name: str) -> str:
...     return 'Hello ' + name
```

https://www.python.org/dev/peps/pep-0484/
https://www.python.org/dev/peps/pep-0526/
https://www.python.org/dev/peps/pep-3107/

Magic аттрибуты функций

```
>>> def foo():
...     print('Foo called!')
...
>>> foo()
Foo called!
>>> dir(foo)
['__annotations__', '__call__', '__class__', '__closure__', '__code__',
'__defaults__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__get__', '__getattribute__', '__globals__',
'__gt__', '__hash__', '__init__', '__kwdefaults__', '__le__', '__lt__',
'__module__', '__name__', '__ne__', '__new__', '__qualname__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__']
>>> foo.__name__
'foo'
```

Объявление функций: резюме

- Функция - это объект
- Функции в Python могут принимать произвольное количество позиционных и ключевых аргументов.
- Для объявления таких функций используют синтаксис упаковки, а для вызова синтаксис распаковки

```
>>> def f(*args, **kwargs):
...     pass
...
>>> f(1, 2, 3, **{"foo": 42})
```

Built-in функции

abs(x)	Return the absolute value of a number
all(*iterable*)	Return True if all elements of the *iterable* are true (or if the iterable is empty)
ord(c)	Given a string representing one Unicode character, return an integer representing the Unicode code point of that character.
chr(*i*)	Return the string representing a character whose Unicode code point is the integer *i*.
input([*prompt*])	The function then reads a line from input, converts it to a string

https://docs.python.org/3/library/functions.html

Task #1

Дописать универсальные функции работы с множествами

```
def intersect(*args):
    pass
```

```
def union(*args):
    pass
```

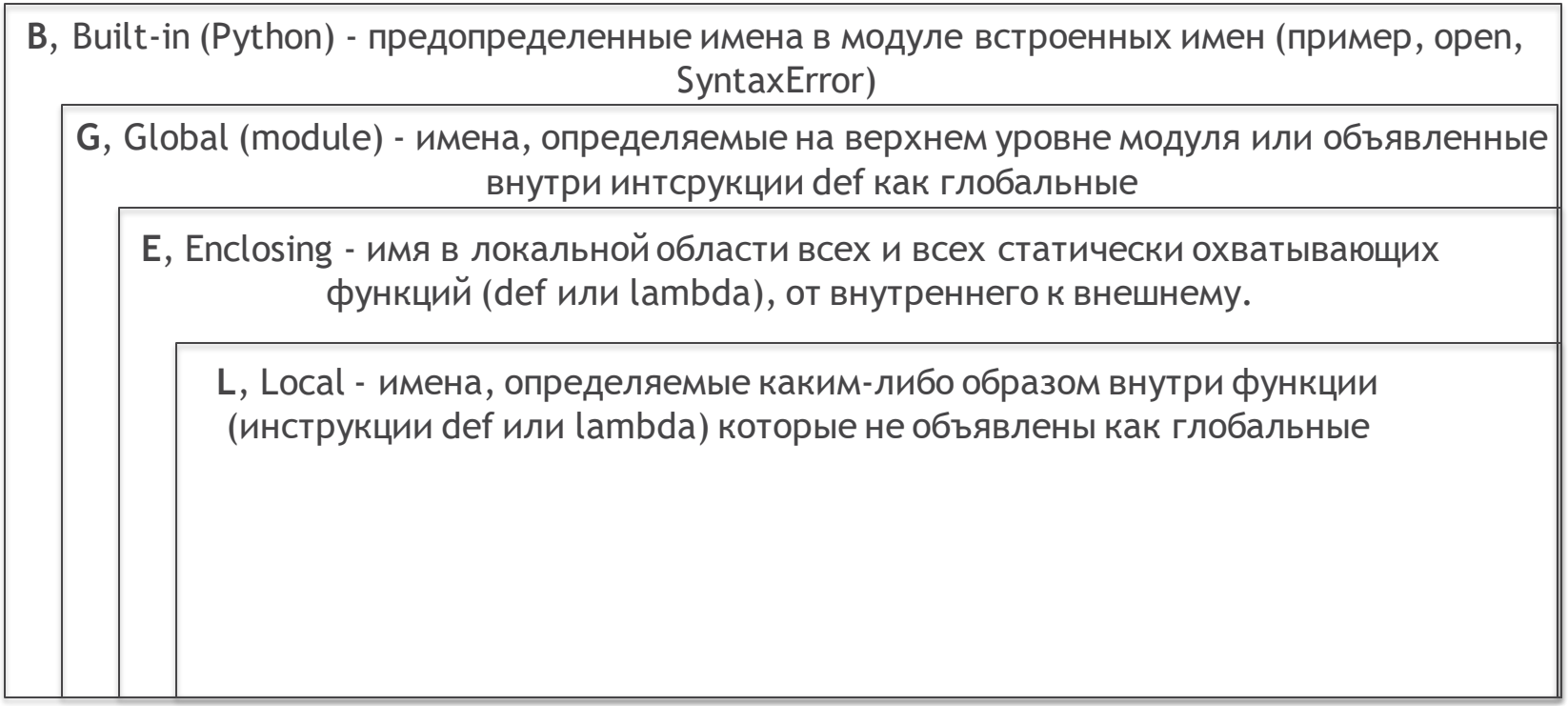
```
>>> intersect(s1, s2), union(s1, s2) # Два операнда
(['S', 'A', 'M'], ['S', 'P', 'A', 'M', 'C'])
>>> intersect([1,2,3], (1,4)) # Смешивание типов
[1]
```

Области видимости
aka scopes

Правила видимости имен

- Объемлющий модуль - это глобальная область видимости.
- Глобальная область видимости охватывает единственный файл.
- Каждый вызов функции создает новую локальную область видимости.
- Операция присваивания создает локальные имена, если они не были объявлены global или nonlocal.

Области видимости: LEGB



Встроенная область видимости

```
>>>import builtins
>>>dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BufferError', 'BytesWarning', 'DeprecationWarning', 'EOFError', 'Ellipsis',
...'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set',
'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super',
'tuple', 'type', 'vars', 'zip']
```

Области видимости: замыкания (фабричные функции)

- Функции в Python могут использовать переменные, определенные во внешних областях видимости.
- Важно помнить, что поиск переменных осуществляется во время исполнения функции, а не во время её объявления.

Области видимости: замыкания (фабричные функции)

```
>>> def wrapper(k) -> None:
...     def inner(n):
...         return n ** k
...
>>> cube = wrapper(3)
>>> cube(2)
8
```

Task #2

Написать функцию, которая обертывает любую, передаваемую в нее функцию и возвращает Tuple[имя функции, результат вызова]

```
>>> d = deanon(int)
>>> d ("123")
('int', 123)
```

```
def deanon(func: callable) -> callable:
    def wrapper(*args, **kwargs):
        return func.__name__, func(*args, **kwargs)
    return wrapper
```

Области видимости: присваивание

- Для присваивания правило LEGB не работает

```
>>> min = 42
>>> def f():
...     return min + 1
...
>>> f()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
File "<stdin>", line 2, in f
UnboundLocalError: local variable 'min' referenced [...]
```

- По умолчанию операция присваивания создаёт локальную переменную.
- Изменить это поведение можно с помощью операторов global и nonlocal.

Оператор global

Позволяет модифицировать значение переменной из глобальной области видимости

```
>>> min = 42
>>> def f():
...     global min
...     min += 1
...     return min
...
>>> f()
43
>>> f()
44
```

Использование global УЖАСНО

Глобальные переменные: подводные камни

```
>>> X = 99
>>> def func1():
...     global X
...     X = 88
...
>>> def func2():
...     global X
...     X = 77
```

Каково значение X?

Области видимости: интроспекция

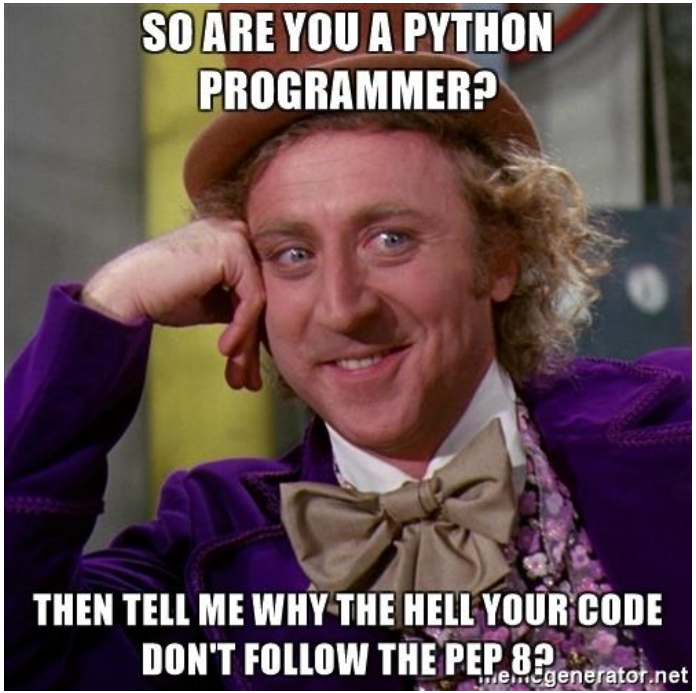
```
>>> min = 42 # globals()["min"] = 42
>>> globals()
{..., 'min': 42}
>>> def f():
...     min = 2 # locals()["min"] = 2
...     print(locals())
...
>>> f()
{'min': 2}
```

Области видимости: резюме

- В Python четыре области видимости: встроенная, глобальная, объемлющая и локальная.
- Правило LEGB: поиск имени осуществляется от локальной к встроенной.
- При использовании операции присваивания имя считается локальным. Это поведение можно изменить с помощью операторов global и nonlocal.

Style Guide

PEP8



PEP 8 содержит стилистические рекомендации по оформлению кода на Python.
Базовые рекомендации:

- 4 пробела для отступов;
- длина строки не более 79 символов для кода и не более 72 символов для документации и комментариев;
- lower_case_with_underscores для переменных и имен функций, UPPER_CASE_WITH_UNDERSCORES для констант.

https://www.python.org/dev/peps/pep-0008/

PEP8: Выражения

- Унарные операции без пробелов, бинарные с одним пробелом с обеих сторон

```
exp = -1.05
value = (item_value / item_count) * offset / exp
```

- Для списков или кортежей с большим количеством элементов используйте перенос с разу после запятой

```
items = [
    'this is the first',
    'set of items',
    'with more items',
    'to come in this line',
    'like this',
]
```

PEP8: Операторы

- Не пишите тело оператора на одной строке с самим оператором

```
if bar: x += 1 # Плохо
while bar: do_something() # Плохо

if bar: # Лучше
    x += 1
while bar: # Лучше
    do_something()
```

- Не используйте Йода-сравнения в if и while

```
if 'md5' == method: # Плохо
    pass

if method == 'md5' # Лучше
    pass
```

PEP8: Операторы

Для сравнения на равенство

- объектов используйте операторы == и !=,
- синглтонов используйте is и is not,
- булевых значений используйте сам объект или оператор not, например

```
if foo:
    # ...
while not bar:
    # ...
```

- Проверяйте отсутствие элемента в словаре с помощью оператора not in

```
if not key in d: # Плохо
if key not in d: # Лучше
```

PEP8: Функции

- Не используйте пробелы до или после скобок при объявлении и вызове функции

```
def foo (x, y): # Плохо
    pass

foo( 42, 24 ) # Плохо
```

- Разделяйте определения функций двумя пустыми строками.

Документация

Документирование функций

```
>>> def min(*args): # type(args) == tuple.
...     """
...     """
...     res = float("inf")
...     for arg in args:
...         if arg < res:
...             res = arg
...     return res

>>> min.__doc__
'With zero arguments return infinity. With two or more arguments, return the
smallest argument.'
```

https://www.python.org/dev/peps/pep-0257/

Комментирование

```
# Define sharks variable as a list of strings
sharks = [
    'hammerhead',
    'great white',
    'dogfish',
    'frilled',
    'bullhead',
    'requiem'
]
# For loop that iterates over sharks list and prints each string item
for shark in sharks:
    print(shark)
```

Sphinx

```
def public_fn_with_sphinxy_docstring(name, state=None):
    """This function does something.

    :param name: The name to use.
    :type name: str.
    :param state: Current state to be in.
    :type state: bool.
    :returns: int -- the return code.
    :raises: AttributeError, KeyError
    """
    return 0
```

http://www.sphinx-doc.org/en/master/

Links

https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt#docstring-standard
http://www.sphinx-doc.org/en/master/
https://google.github.io/styleguide/pyguide.html

HW #1

Есть список словарей:

```
friends = [
    {'name': 'Сэм', 'gender': 'Мужской', 'sport': 'Баскетбол'},
    {'name': 'Эмили', 'gender': 'Женский' 'sport': 'Волейбол'},
    ...
]
```

Необходимо создать функции select(*field_name: list) -> None, field_filter(field_name: str, *collection: list) -> None, query(*collection, select, field_filter, ...) -> list.

Пример работы программы:

```
result = query(
    friends,
    select('name', 'gender', 'sport'),
    field_filter('sport', ['Баскетбол', 'Волейбол']),
    field_filter('gender', ['Мужской']),
)
```