План занятия

- "Составные" типы данных
- Сложность операций с "составными" типами данных
- Функции
- Область видимости переменных

Составные типы данных

Составной = из нескольких частей

Кортеж

- Представляет собой набор неизменяемых значений

```
(1, 2)
(1, 2, 3)
('a', 'b', 'c', 'd', 'e', 'f')
```

Практика

Файл "types/tuples.py"

Список

- Представляет собой изменяемый набор значений

```
[1, 2, 1, 3]
[True, False]
['person', 'manager', 'student']
```

Практика

Файл "types/lists.py"

Мы можем добавить новые значения в список.

А в кортеж?

```
numbers = [1, 2, 3]
numbers.append(4)
print(numbers)  # [1, 2, 3, 4]
```

В кортеж - нельзя ничего добавить!

Можно ли изменить объект?

- Если да, то он изменяемый
- Если нет, то он неизменяемый

Практика

Файл "tuple_vs_list.py"

Множество

- Представляет собой изменяемый набор уникальных значений

```
{1, 2}
{'my', 'unique', 'words'}
{'a', False, None, 1}
```

Практика

Файл "types/sets.py"

Словарь

- Представляет собой набор пар {"ключ": "значение"}

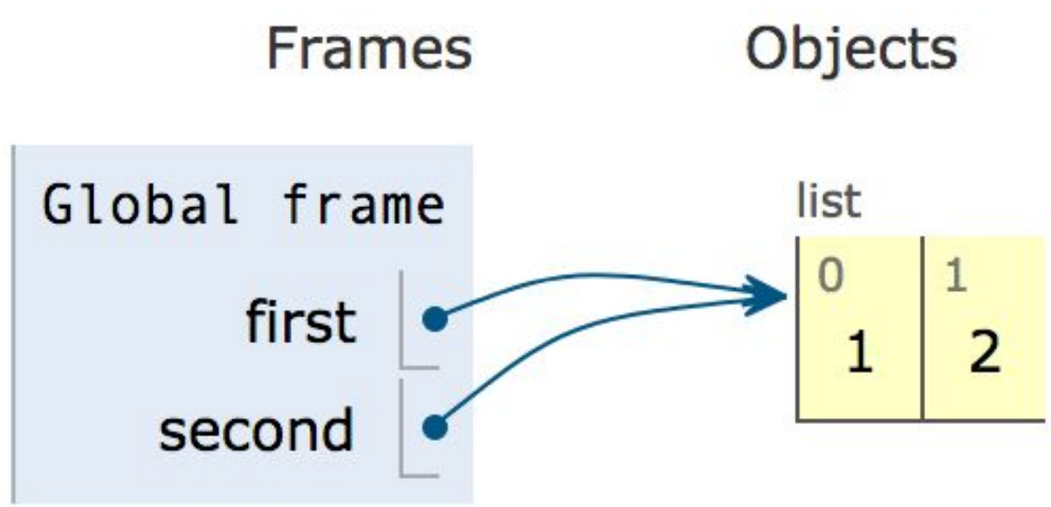
```
{}
{'a': 1, 'b': 2, 'c': 3}
{'name': 'Ivan', 'age': 25, 'working': True}
```

Практика

Файл "types/dicts.py"

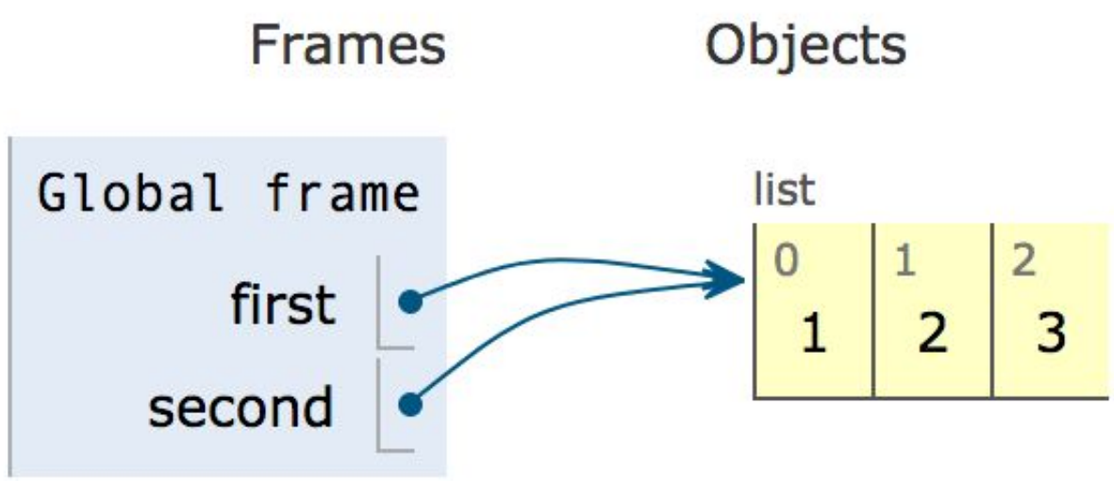
Присваивание никогда не копирует данные

```
first = [1, 2]
second = first
```




```
first.append(3)


print(first)  # [1, 2, 3]
print(second)  # [1, 2, 3]
```



Какова сложность операций?

https://wiki.python.org/moin/TimeComplexity

Каждый тип данных хорош в чем-то своем

- В tuple быстро брать значения по индексу
- В list быстро добавлять новое, брать значения по индексу
- В set быстро искать значения, делать операции над множествами
- В dict быстро брать значения по ключу, искать ключи, добавлять новые

Промежуточные итоги

Неизменяемые типы (immutable):

- Простые типы: int, float, complex, bool, str, None
- Кортежи: tuple

Изменяемые типы (mutable):

- Списки: list
- Множества: set
- Словари: dict

Функции

Вызов функции

- Уже знакомая для нас конструкция:

```
len('abc')
print('2 + 2 = ', 4)
input('How are you?')
```

Определение функции

Определение функции

```
def sum_two_numbers(number1, number2):
    ...
```

Где:

`def` - ключевое слово для объявления функции
`sum_two_numbers` - имя функции
`number1, number2` - входные параметры

Определение функции

```
def sum_two_numbers(input_var1, input_var2):
    return input_var1 + input_var2
```

Где:

`return` - ключевое слово, чтобы вернуть значение
`returns input_var1 + input_var2` - тело функции

Определение и вызов функции неразрывно связаны!

Практика

Файл "functions/definition.py"

Функция может принимать

- Нулевое количество аргументов: **print**() #
- Позиционные аргументы: **print**(1, 2) *# 1 2*
- Именованные аргументы: **print**(1, 2, sep=',') *# 1,2*

Функция может принимать

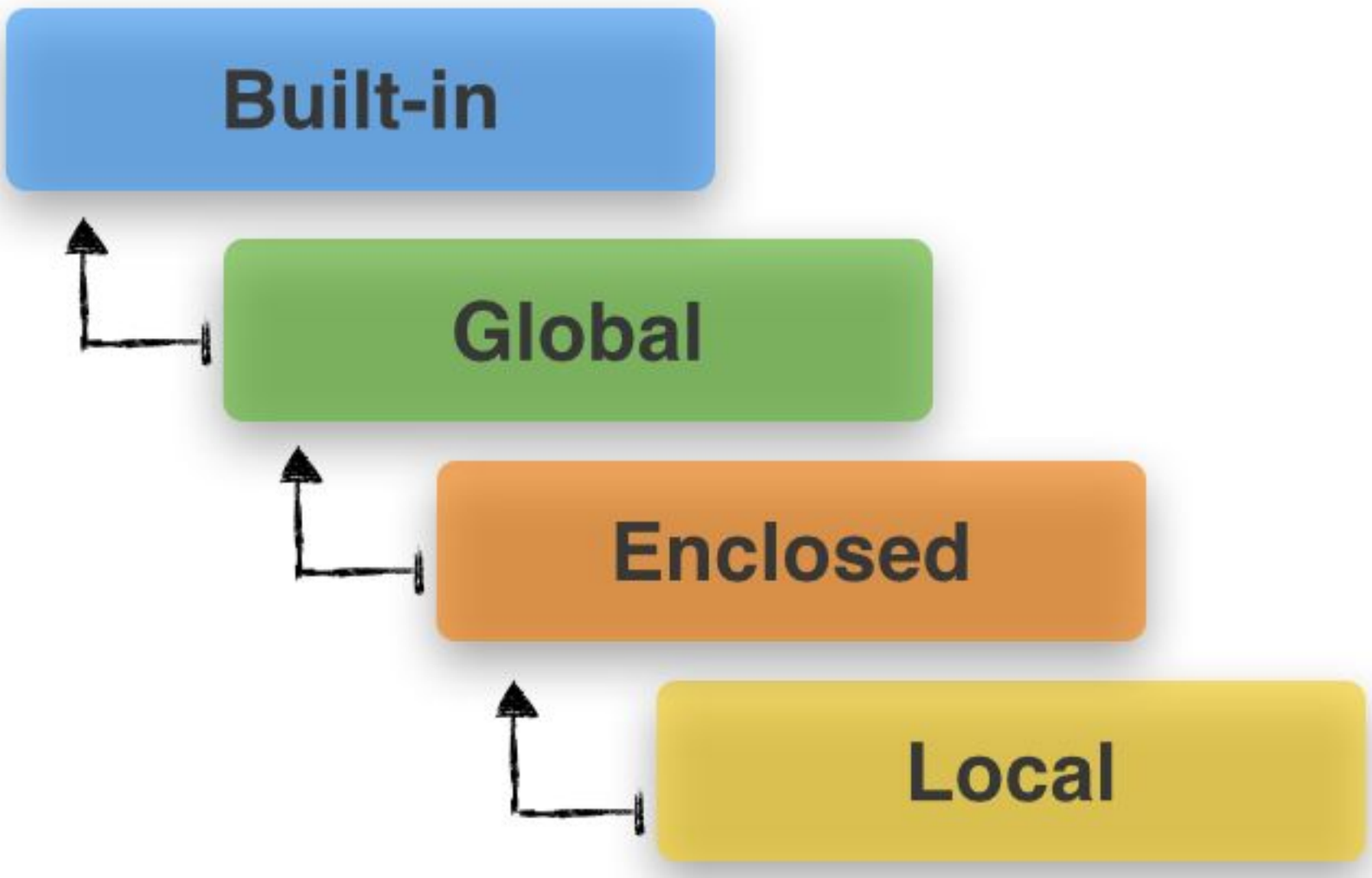
- Любое количество позиционных аргументов:
  print(1, 2, 3, 4, 5, 6, 7, 8, 9, ...)

- Любое количество именованных аргументов:
  dict(name='Ivan', age=25, working=**True**)

Практика

Файл "functions/args_and_kwargs.py"

Области видимости



Области видимости

```
LANGUAGE_NAME = 'Python'

def print_language_name():
    heart = '<3'
    print('I am learning', LANGUAGE_NAME, heart)

print(LANGUAGE_NAME, 'is cool!') # no heart ;(
```

Области видимости нужны нам, чтобы прятать лишнее!

Практика

Файл "scopes.py"

Выводы

- Узнали про "составные" типы данных
- Поняли, что разные операции могут затрачивать разное количество времени работы. Нужно с умом выбирать нужный тип данных!
- Научились вызывать и определять функции
- Поняли, как прятать лишние переменные