15-110 Principles of Computing – F19

Lecture 10:
Tuples, Lists 1

Teacher:
Gianni A. Di Caro

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

So far about Python ...

- Basic elements of a program:
  - Literal objects
  - Variables objects
  - Function objects
  - Commands
  - Expressions
  - Operators

- Utility functions (built-in):
  - `print(arg1, arg2, …)`
  - `type(obj)`
  - `id(obj)`
  - `int(obj)`
  - `float(obj)`
  - `bool(obj)`
  - `str(obj)`
  - `input(msg)`
  - `len(non_scalar_obj)`

- Object properties
  - Literal vs. Variable
  - Type
  - Scalar vs. Non-scalar
  - Immutable vs. Mutable

- Conditional flow control
  - ```
    if cond_true:
        do_something
    ```
  - ```
    if cond_true:
        do_something
    else:
        do_something_else
    ```
  - ```
    if cond1_true:
        do_something_1
    elif cond2_true:
        do_something_2
    else:
        do_something_else
    ```

- Data types:
  - `int`
  - `float`
  - `bool`
  - `str`
  - `None`

- Relational operators
  - `>`
  - `<`
  - `>=`
  - `<=`
  - `==`
  - `!=`

- Logical operators
  - `and`
  - `or`
  - `not`

- Operators:
  - `=`
  - `+`
  - `+=`
  - `-`
  - `/`
  - `*`
  - `*=`
  - `//`
  - `%`
  - `**`
  - `[]`
  - `[:]`
  - `[::]`

- String methods
    ```

Tuple type

- **Tuple:** (non-scalar type) ordered **sequence** of *objects* (any type, not need same type), **immutable**
- Purpose: ***group*** **items together in the same (immutable) object**
- **Syntax:** `(a,b,c, …, )`
  - Examples of tuple variables assigned with tuple literals

```
prime_numbers = (1, 3, 5, 7, 11)
irrational_numbers = (2.71828, 3.14159, 1.41421)
fruits = ('apple', 'pear', 'banana', 'orange')
colors = ('red', 'blue', 'green')
person_info = ('Donald', 'Trump', 14, 'June', 1946, 'President')
fruit_info = ('melon', 3.6, 'yellow', 12.5)
logical_values = (True, False, True, 255)
empty_sequence = ()
one_element_sequence = (5, )  → one_integer_var = (5)
```

Tuple type

- **Syntax:** `(a,b,c, …, )`
- … but also **any comma-separated sequence of objects**: `a, b, c, d`

```
prime_numbers = 1, 3, 5, 7, 11
irrational_numbers = 2.71828, 3.14159, 1.41421
fruits = 'apple', 'pear', 'banana', 'orange'
colors = 'red', 'blue', 'green'
person_info = 'Donald', 'Trump', 14, 'June', 1946, 'President'
fruit_info = 'melon', 3.6, 'yellow', 12.5
logical_values = True, False, True, 255
one_element_sequence = 5,
```

`a,b,c = 1,3,6` → three variables of `int` type

`a = 1,3,6` → one variable of `tuple` type

List type

- **List:** (non-scalar type) ordered **sequence** of *objects* (any type, not need same type), **mutable**
- Purpose: ***group*** **items together in the same (mutable) object**
- **Syntax:** `[a,b,c, …, ]`

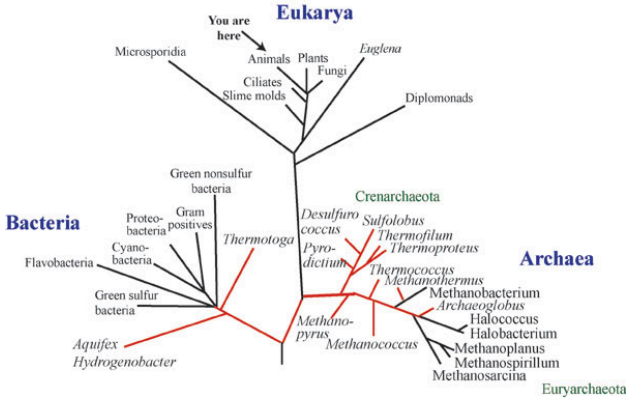
➢ Examples of list variables assigned with list literals

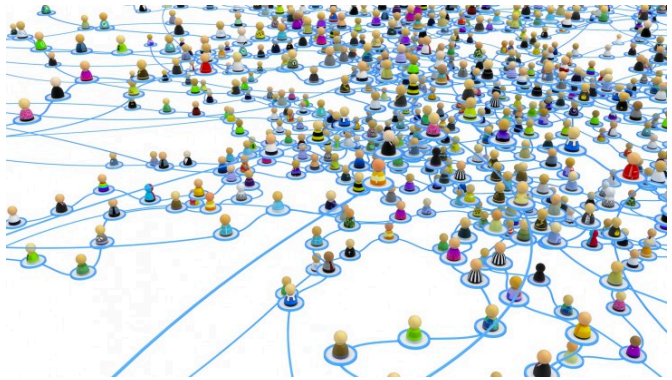
```
prime_numbers = [1, 3, 5, 7, 11]
irrational_numbers = [2.71828, 3.14159, 1.41421]
fruits = ['apple', 'pear', 'banana', 'orange']
person_info = ['Donald', 'Trump', 14, 'June', 1946, 'President']
fruit_info = ['melon', 3.6, 'yellow', 12.5]
logical_values = [True, False, True, 255]
empty_list = []

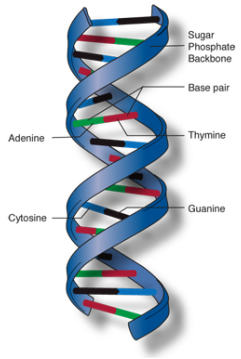
one_element_list = [5]  or  one_element_list = [5,]
list_of_lists = [1, 'sedan', ['Toyota', 'Corolla', 1.8, 2012], 52000]
```

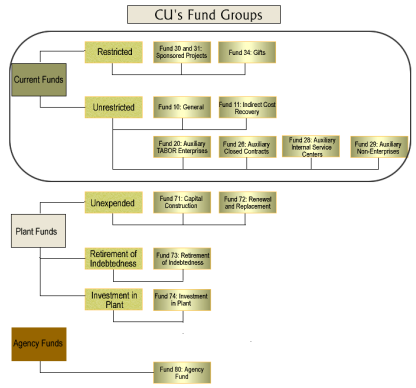
List and tuples: fundamental *data structures* in python

- **Data structure: Container used to to store data (information) in a specific organized form, a *layout***
- A given *layout* determines a data structure to be efficient in some operations and/or effective in the representation of parts of information, and maybe inefficient / ineffective in others
- E.g., the layout affects how we access *access / find / change / add / remove / sort/organize* data



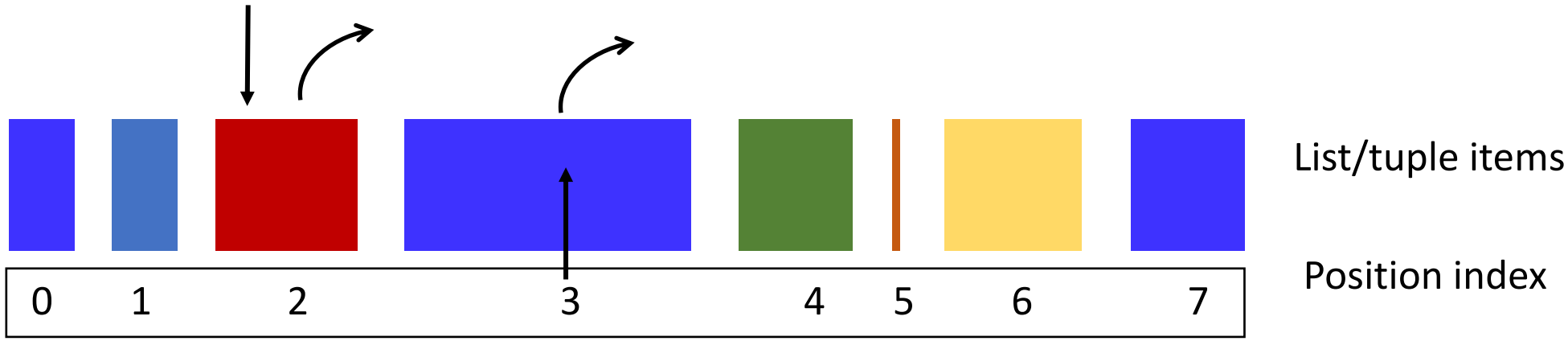






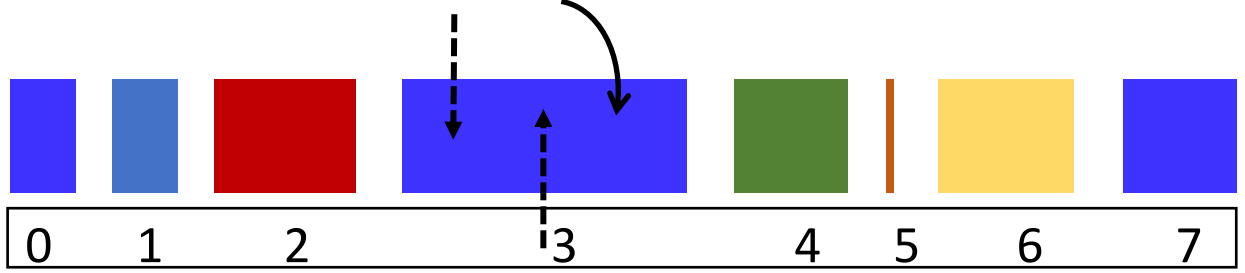
List and tuples data structures: basic operations

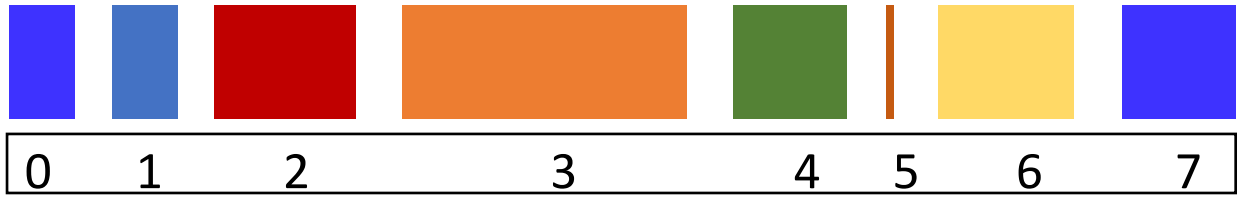
- **Data structure: Container used to to store data (information) in a specific organized form, a *layout***
- List/tuple layout is *linear*: based on a sequential arrangement of the data → **position indexes**
  - **Access (read) operations**, at any position, for both lists and tuples:
    - get data by index (in 2 steps time)
    - find data by content (linear number steps for scanning the list, depending on item's position)

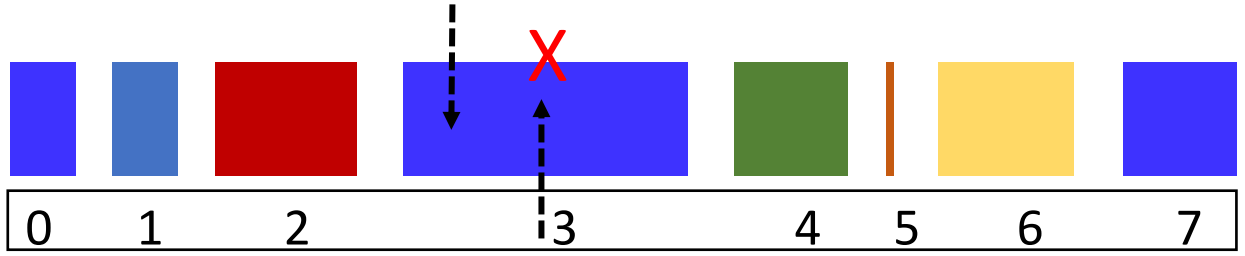


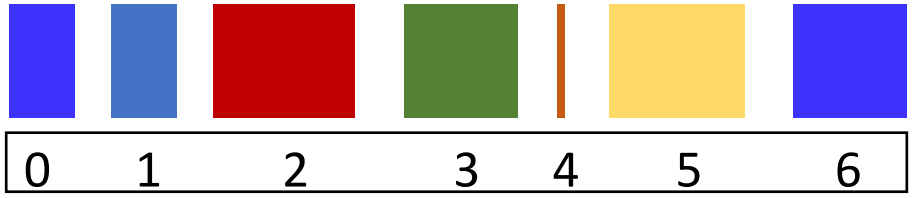
List data structures: basic operations

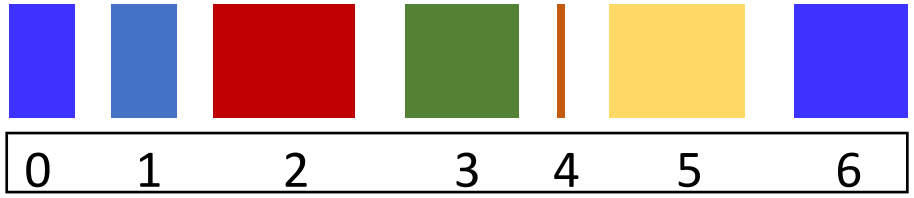
- **Update operations**, *at any position*, only for lists:
  - **change** the value of an item
  - **remove** one item
  - **add** a new item anywhere (at the end, at the beginning, in between)

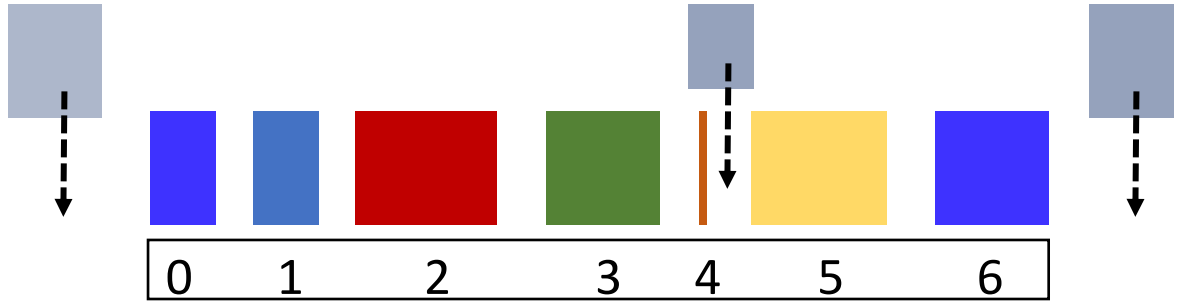




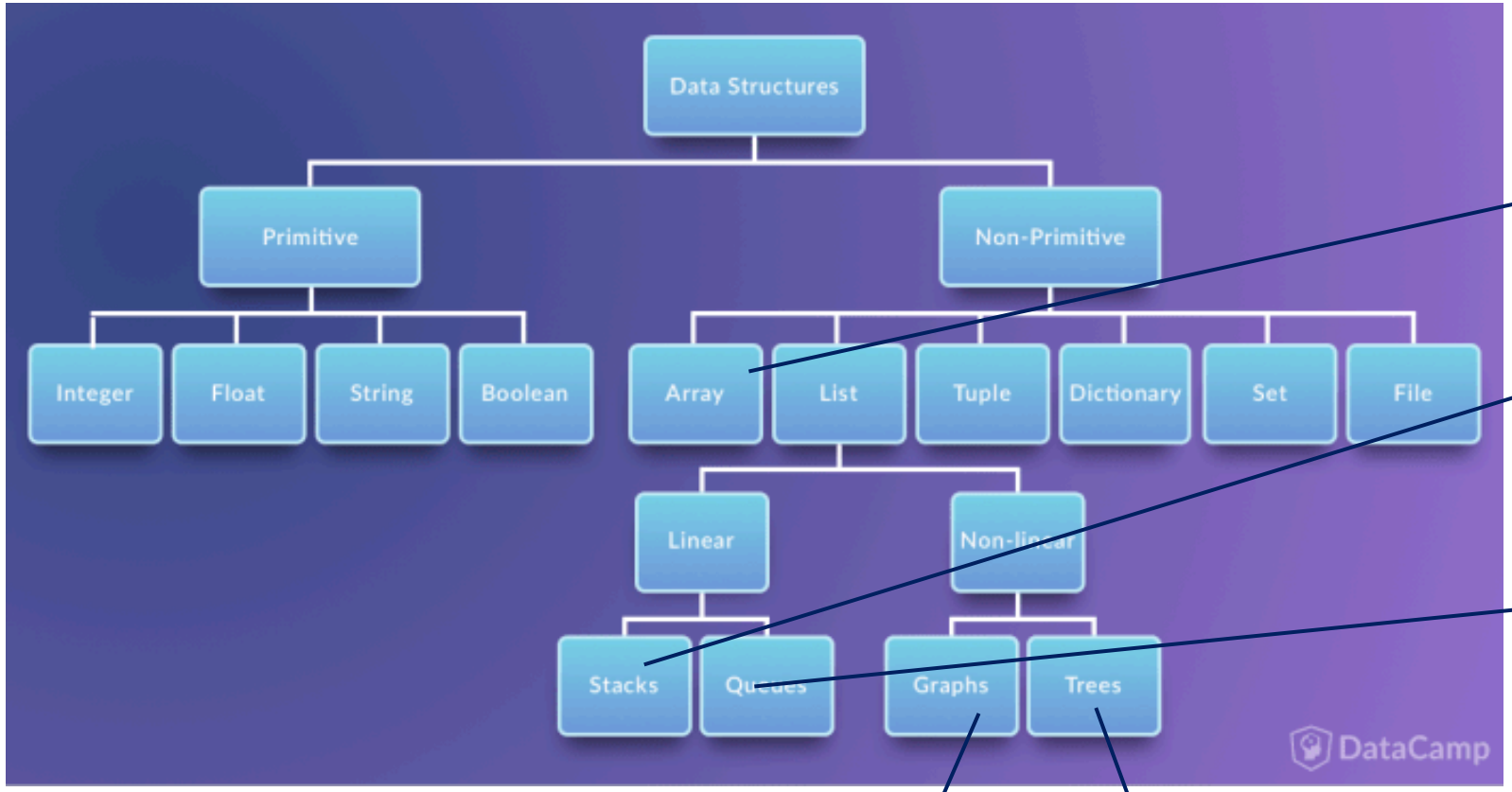


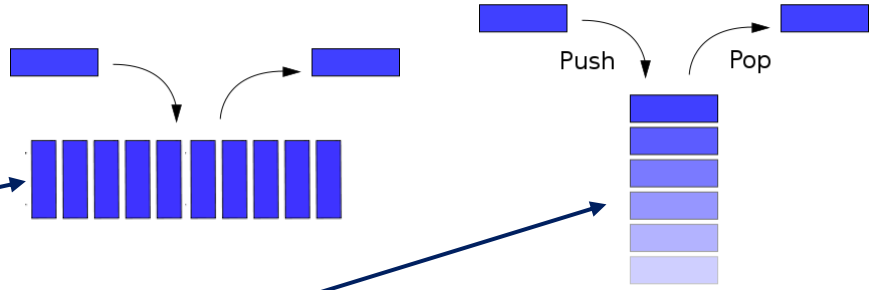


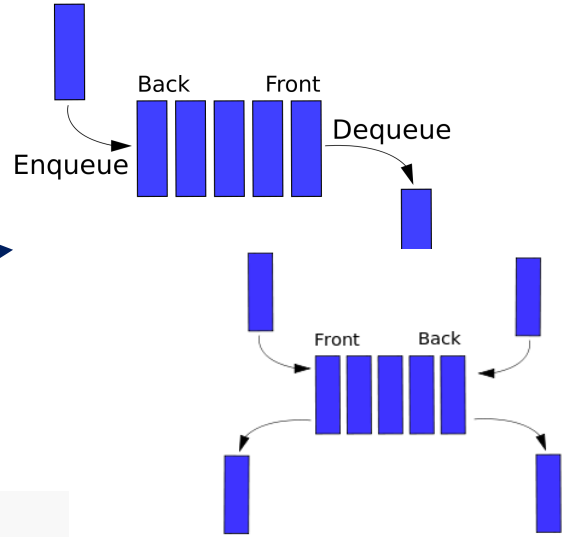


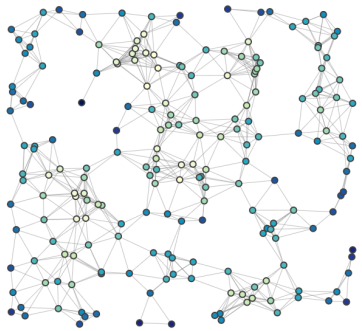


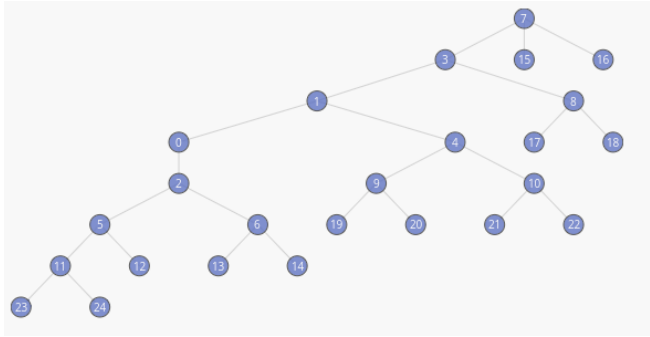
Other data structures in python (building on lists)











Accessing tuple / list values: [], [:], [::]

- **Syntax:** the same used for strings!
  - *Single* tuple/list element: `[index]`
  - *Subsequence* of the tuple/list: `[start : end]`
  - *Subsequence with a stride*: `[start : end : step]`

tuple

```
prime_numbers = (1, 3, 5, 7, 11)
```

list

```
even_numbers = [2, 4, 6, 8, 10]
```

`x = prime_numbers[0]` → `x` is of type `int` and has value 1

`x = even_numbers[2]` → `x` is of type `int` and has value 6

`x = prime_numbers[-3]` → `x` is of type `int` and has value 5

`x = prime_numbers[1:3]` → `x` is of type `tuple` and has value `(3,5)`
corresponding to the tuple elements of index 1 and 2

`x = even_numbers[1:4:2]` → `x` is of type `list` and has value `(3,7)`
corresponding to the list elements of index 4 and 8

Accessing tuple/list values that are inside tuples/lists

```
cars = [1, 'sedan', ['Toyota', 'Corolla', 1.8, 2012], 52000]
```

- How do we access the item `Corolla` in the list variable `cars`?

```
model = cars[2][1]
```

Equivalent to:

```
model = (cars[2])[1]
```

```
['Toyota', 'Corolla', 1.8, 2012]
'Corolla'
```

- How do we access to the 4rd character in the `model` variable? (of type `str`)

Equivalent to:

```
model = ((cars[2])[1])[3]
```

```
model3 = cars[2][1][3]
```


```
['Toyota', 'Corolla', 1.8, 2012]
'Corolla'
'o'
```

Updating tuple values: No!

- **Updating tuple values: No! tuples are immutable types,** like strings
- *Invalid operations:*

```
prime_numbers = (1, 3, 5, 7, 11)
```

`prime_numbers[4] = 13` → an existing element cannot be modified

`prime_numbers[5] = 13` → it cannot *be extended* to an additional element

- "Update" the tuple by creating a new tuple from the existing one

```
prime_numbers = (1, 3, 5, 7, 11)
new_prime_numbers = (1, 3, 5, 7, 13)
new_prime_numbers = prime_numbers[0:4] + (13, )
new_prime_numbers = prime_numbers + (13, 17)
```