파이썬 프로그래밍
강의 노트 #08

자료구조 1
리스트(list), 튜플(tuple)

리스트 (List)

- 리스트는 순서가 있는 요소들의(Sequential Items) 모음
  - **변경 가능**한 자료구조
  - 크기를 동적으로(run-time) 조절 가능
  - 요소를 치환해서 변환 가능
  - 인덱싱을 이용해서 요소를 참조 가능
  - 문자열과 마찬가지로 연결하기, 슬라이싱(slicing), 반복하기, 멤버 검사, 길이 정보 등과 같은 다양한 연산 지원
  - 리스트는 [] 형태로 표현

```
[]    # 빈 리스트(empty list)
[1, 2, 3, "abc"] # 요소가 4개인 리스트
[1, 2, 3, [1, 2]] # 요소에 리스트가 포함됨
```

리스트 생성

- 빈 리스트 생성 후 요소 추가
- 빈 리스트 생성

```
변수_이름 = []
변수_이름 = list()
```

- 리스트 요소 추가
  - append() 함수는 리스트의 마지막에 새로운 요소들을 추가

```
a = []
a.append(4)        # [4]
a.append('5')      # [4, '5']
a.append([2, 3])   # [4, '5', [2, 3]]
```

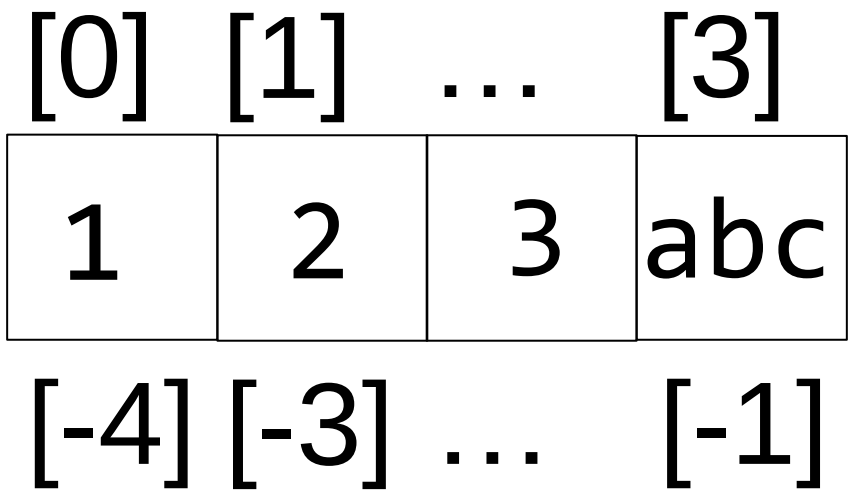
리스트 (List)

- 리스트의 요소 개수 확인하기
  - len() 함수 사용

```
e = []    # empty list
l1 = [1, 2, 3, "abc"] # 요소가 4개인 리스트
l2 = [1, 2, 3, [1, 2]] # 요소에 리스트가 포함됨
len([])   # 0
len(e)
len(l1)
len(l2)
len([1, 2, 3, "abc"])
len([1, 2, 3, [1, 2]])
```

리스트 (List)

- 인덱싱
  - 특정(단일) 요소에 접근(access)해서 값을 읽거나 변경하는 방법
  - **[인덱스]**를 사용해서 요소에 접근

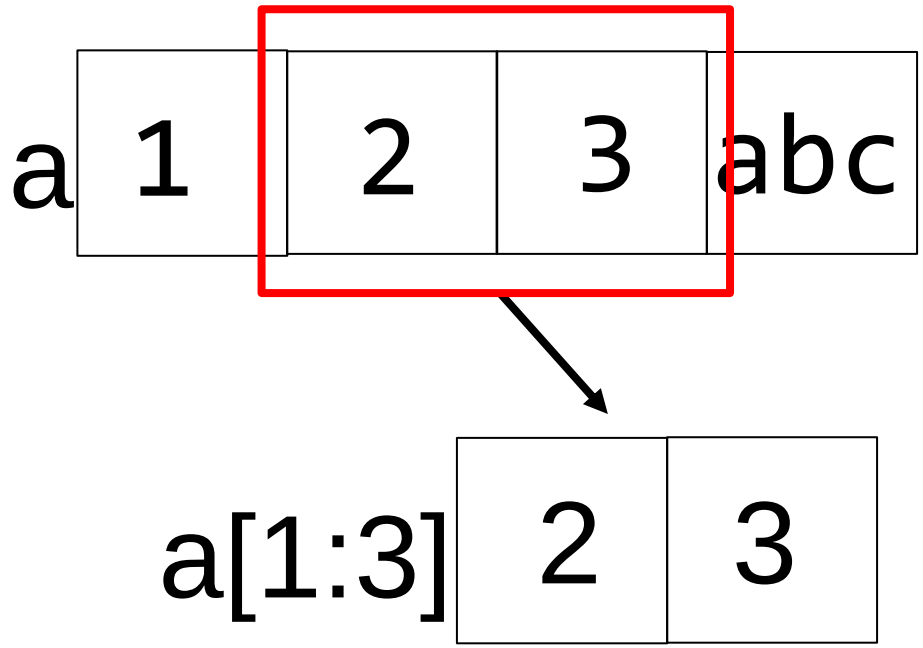



```
l = [1, 2, 3, "abc"] # 리스트 생성
print(l[0], l[1], l[2], l[3]) # 순서대로 출력
print(l[-1], l[-2], l[-3], l[-4]) # 거꾸로 출력
```

리스트 (List)

- 슬라이싱(slicing)
  - 여러 개 요소에 접근하는 방법
  - **[시작인덱스:끝인덱스]**를 사용해서 요소들에 접근
  - 시작인덱스에 해당되는 요소부터 끝인덱스 - 1 까지의 요소들을 포함하는 리스트 참조

```
a = [1, 2, 3, "abc"]
a[1:3]
```



리스트 (List)

- 슬라이싱(slicing)

```
lst1 = [1, 2, 3, "abc", 4, 5]
lst1[2:4]     # [3, "abc"]
lst1[:3]      # [1, 2, 3]
lst1[2:]      # [3, "abc", 4, 5]
lst1[:]       # [1, 2, 3, "abc", 4, 5]
```

- 확장 슬라이싱
  - [시작인덱스:끝인덱스:스텝]
  - 시작인덱스부터 끝인덱스 – 1까지 스텝만큼 건너뛰면서 리스트 참조

```
lst1 = [1, 2, 3, "abc", 4, 5]
lst1[2:5:2]     # [3, 4]
lst1[2:6:2]     # [3, 4]
lst1[:3:2]      # [1, 3]
lst1[2::2]      # [3, 4]
```

리스트 (List)

- 연결하기
  - + 연산자를 이용해서 두 개 이상의 리스트를 연결해서 새로운 리스트를 생성할 수 있음

```
a = [1, 2, 3]
b = ["abc", 4, 5]
c = a + b # 두 개의 리스트를 붙여서 c에 저장
print(c)  # [1, 2, 3, "abc", 4, 5]
```

- 반복하기
  - * 연산자를 이용해서 요소들이 반복되는 리스트 생성

```
a = [1, 2]
b = a * 4
print(b)      # [1, 2, 1, 2, 1, 2, 1, 2]
```

리스트 (List)

- 리스트 요소 수정
  - 특정 요소의 값을 변경
    - **리스트이름[인덱스] = 새로운 값**

```
a = [1, 2, 3, "abc"]
a[2] = 4      # [1, 2, 4, "abc"]
a[3] = ["ab", 'a', "def"] # [1,2,4,["ab",'a',"def"]]
```

  - 여러 개 요소를 새로운 값으로 치환하고자 할 때에는 아래와 같이 슬라이싱 방법을 사용해서 수정
  - **리스트이름[시작인덱스:끝인덱스] = 새로운 값**

```
a = [1, 2, 3, "abc"]
a[1:2] = ['g', 'h'] # [1, 'g', 'h', 3, "abc"]
a[2:4] = ["ab", 'a', "def"]
# [1, 'g', "ab", 'a', "def", "abc"]
```

리스트 (List)

- 리스트 요소 삭제
  - del 명령을 이용해서 특정 요소 삭제
  - del 삭제하고 싶은 요소 또는 del(삭제하고 싶은 요소)

```
a = [1, 2, 3, "abc", 'g', 'h']
del a[1] # a[1:2] = [], [1, 3, "abc", 'g', 'h']
```

  - 범위 삭제
  - **리스트이름[시작인덱스:끝인덱스] = []**

```
a = [1, 2, 3, "abc", 'g', 'h']
a[1:2] = [] # [1, 3, "abc", 'g', 'h']
a[2:4] = [] # [1, 3, 'h']
del(a[0:2]) # ['h']
```

리스트 (List)

- 리스트 요소 제거
  - 리스트의 요소를 제거하려면 remove(x) 활용
  - 리스트에서 첫 번째로 나오는 x 제거

```
a = [1, 3, 5, 7, 9, 7, 5, 3, 1]
a.remove(5)   # [1, 3, 7, 9, 7, 5, 3, 1]
a.remove(3)   # [1, 7, 9, 7, 5, 3, 1]
a.remove(3)   # [1, 7, 9, 7, 5, 1]
```

- 리스트에 포함된 요소 x의 개수 세기
  - count(x) 함수 활용
  - 리스트에서 x의 개수 반환

```
a = [1, 5, 7, 5, 7, 5, 3, 1]
a.count(5)                # 3
a.count(3)                # 1
a.count(7)                # 2
```

리스트 (List)

- 요소의 존재 여부 확인하기
  - in 사용

```
a = [4, 8, 7, 2, 1]
8 in a          # True
9 in a          # False
```

- 인덱스 확인하기

```
a = [4, 8, 7, 2, 1]
a.index(8)          # 1
a.index(1)          # 4
```

리스트 (List)

- 리스트에 요소 삽입
  - **요소 삽입은 insert(x, y)함수 활용. x index 위치에 y 삽입**

```
a = [4, 8, 7, 2, 1]
a.insert(2, 5)  # [4, 8, 5, 7, 2, 1]
a.insert(1, [1, 6])
# [4, [1, 6], 8, 5, 7, 2, 1]
```

리스트 (List)

- 리스트 요소 정렬(sort)
  - sort()함수 활용

```
a = [4, 8, 7, 2, 1]
a.sort()       # [1, 2, 4, 7, 8]
a.append('a') # [1, 2, 4, 7, 8, 'a']
a.sort() # Error 발생: 문자열과 정수를 비교함

b = ['g', 'o', 'o', 'd']
b.sort()       # ['d', 'g', 'o', 'o']
```

- 리스트 요소 뒤집기
  - reverse()함수 활용

```
a = [4, 8, 7, 2, 1]
a.reverse()              # [1, 2, 7, 8, 4]
```

for 반복문과 리스트 사용하기

- 리스트는 순서가 있는 객체
- for 반복문을 이용해서 리스트의 각 요소에 접근 가능

```
a = [4, 8, 7, 2, 1]
for n in a:
    print(n)
l = [1, 'g', 'h', 3, "abc"]
for n in l:
    print(n)

# 리스트에서 4보다 큰 정수만 출력
for n in a:
    if n > 4:
        print(n)
```

실습문제 1

- 문제
  - 정수 한 개(n)를 함수의 인자로 받고, 그 정수의 n개만큼 사용자로부터 정수를 입력 받고, 이를 요소로 리스트를 구성한 후 함수의 결과값으로 반환하는 함수 작성. 이 함수를 이용해서 5개의 정수 값으로 구성된 리스트를 생성하고 출력하는 프로그램 작성
- 요구사항
  - 사용자로부터 입력받을 정수의 개수는 함수에 인자로 전달
  - 사용자로부터 입력받는 값은 양의 정수로 가정
  - 함수를 검수할 때 입력 받아서 리스트로 만들 정수는 5개

실습문제 1

- 최종 코드

```
def createListFromInput(n):
    lst = []
    for i in range(n):
        num = int(input("정수를 입력하세요: "))
        lst.append(num)
    return lst

lst1 = createListFromInput(5)
print(lst1) # 리스트를 확인차 출력

for n in lst1:
    print(n) # 요소 출력
```

실습문제 2

- 문제
  - 정수 한 개(n)를 함수의 인자로 받고, 그 정수의 모든 약수를 리스트로 구성해서 반환하는 함수(createDivisorsList)를 구현. 이 함수를 이용해서 사용자로부터 1~1000 정수 중에서 하나를 입력 받고, 그것의 모든 약수들의 합을 계산해서 출력하는 프로그램 작성
  - 약수
    - 정수 n의 약수를 찾는 방법은 1~n까지의 정수를 n으로 나눠서 나머지가 0인지 확인
- 요구사항
  - 사용자로부터 입력 받는 값은 1~1000 정수로 가정
  - createDivisorsList 함수를 호출하는 코드에서 리스트의 내용을 화면에 출력함

실습문제 2

- 최종 코드

```
def createDivisorsList(n):
    lst = []
    for i in range(1, n + 1):
        if n % i == 0:
            lst.append(i)
    return lst

# 사용자로부터 정수 입력 받기
n = int(input("1000이하의 정수 한 개를 입력하세요: "))
```

실습문제 2

```
# 약수 리스트 생성
lst = createDivisorsList(n)
print(lst) # 약수 리스트 확인을 위해 화면에 출력

# 약수의 합 구하기
sum = 0
for i in lst:
  sum += i
print(f"약수들의 총 합: {sum}") # 합계 출력
```

튜플 (Tuple)

- 튜플은 리스트와 비슷하게 순서가 있는 요소들의 (Sequential Items) 목록
  - **변경할 수 없는** 자료구조 (리스트와의 큰 차이)
  - 리스트와 마찬가지로 연결하기, 슬라이싱(slicing), 반복하기, 멤버 검사, 길이 정보 등과 같은 다양한 연산 지원 (새로운 튜플 생성 가능)
  - () 형태로 표현 (괄호를 생략하는 경우도 있음)

```
()   # 빈 튜플(empty tuple)
(1, 2, 3, "abc") # 요소가 4개인 튜플
(1, 2, 3, (1, 2)) # 요소에 튜플 포함 가능
(1, 2, 3, [1, 2]) # 요소에 리스트 포함 가능
tuple1 = (2, 3)
tuple2 = 2, 3    # 괄호 생략 가능
tuple3 = 2, 3, [1, 2], (1, 2)
```

튜플 (Tuple)

- 요소가 한 개인 튜플을 생성할 때에는 반드시 콤마(',')로 끝내야 함

```
tuple1 = (2,)         # 튜플 생성
tuple2 = 2,           # 튜플 생성
tuple3 = ("hello",) # 튜플
tuple4 = "hello",   # 튜플
tuple5 = (2)          # 정수
tuple6 = ("hello")  # 문자열
```

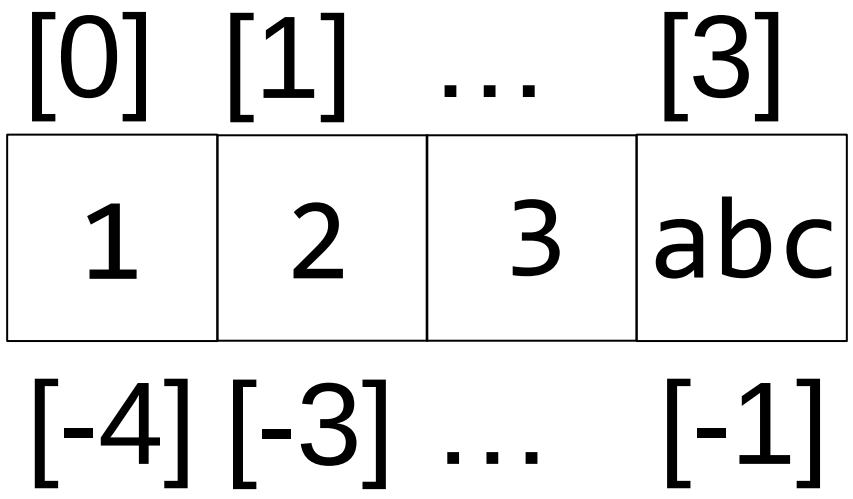
튜플 (Tuple)

- 튜플의 요소 개수 확인하기
  - len() 함수 사용

```
e = ()
t1 = (1, 2, 3, "abc")
t2 = (1, 2, 3, (1, 2))
t3 = 1, 2, 3, [1, 2], (1, 2)
len(())   # 0
len(e)
len(t1)
len(t2)
len()
len((1, 2, 3, "abc"))
len((1, 2, 3, [1, 2]))
```

튜플 (Tuple)

- 인덱싱
  - 특정(단일) 요소에 접근(access)해서 값을 읽거나 변경하는 방법
  - **[인덱스]**를 사용해서 요소에 접근

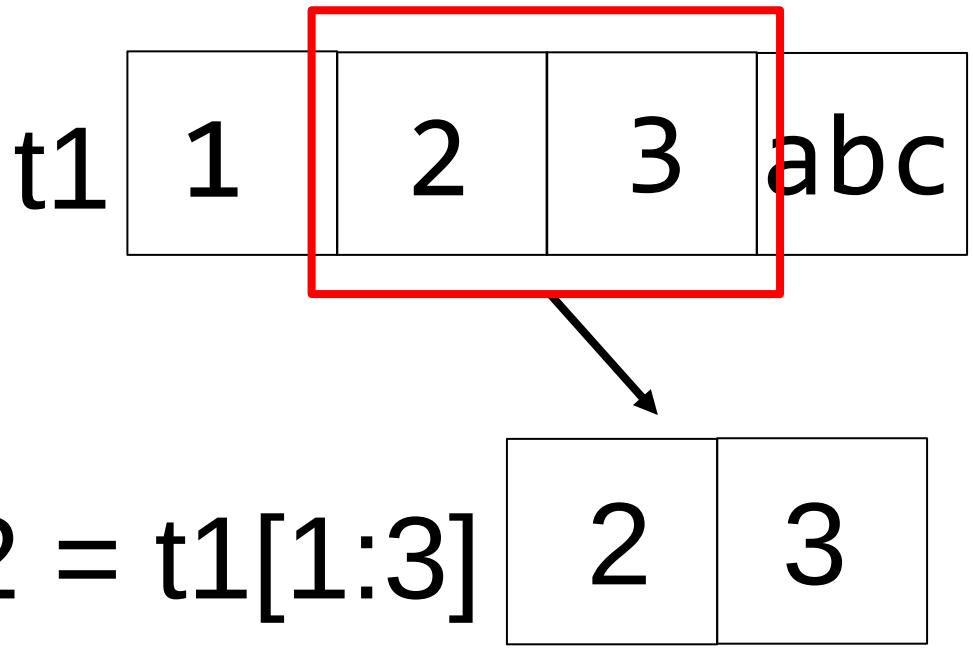



```
a = (1, 2, 3, "abc") # 튜플 생성
print(a[0], a[1], a[2], a[3]) # 순서대로 출력
print(a[-1], a[-2], a[-3], a[-4]) # 거꾸로 출력
```

튜플 (Tuple)

- 슬라이싱(slicing)
  - 여러 개 요소에 접근하는 방법
  - **[시작인덱스:끝인덱스]**를 사용해서 요소들에 접근
  - 시작인덱스에 해당되는 요소부터 끝인덱스 - 1 까지의 요소들을 포함하는 튜플 생성

```
t1 = (1, 2, 3, "abc")
t2 = t1[1:3]
```



튜플 (Tuple)

- 슬라이싱(slicing)

```
a = (1, 2, 3, "abc", 4, 5)
a[2:4]     # (3, "abc")
a[:3]      # (1, 2, 3)
a[2:]      # (3, "abc", 4, 5)
a[:]       # (1, 2, 3, "abc", 4, 5)
```

- 확장 슬라이싱
  - [시작인덱스:끝인덱스:스텝]
  - 시작인덱스부터 끝인덱스 – 1까지 스텝만큼 건너뛰면서 튜플 참조

```
a = (1, 2, 3, "abc", 4, 5)
a[2:5:2]     # (3, 4)
a[2:6:2]     # (3, 4)
a[:3:2]      # (1, 3)
a[2::2]      # (3, 4)
```

튜플 (Tuple)

- 연결하기
  - + 연산자를 이용해서 두 개 이상의 튜플을 연결해서 새로운 튜플을 생성할 수 있음

```
t1 = (1, 2, 3)
t2 = ("abc", 4, 5)
t = t1 + t2 # 두 개 튜플을 붙여서 t에 저장
print(t)  # (1, 2, 3, "abc", 4, 5)
```

- 반복하기
  - * 연산자를 이용해서 요소들이 반복되는 튜플 생성

```
a = (1, 2)
b = a * 4
print(b)      # (1, 2, 1, 2, 1, 2, 1, 2)
```

튜플 (Tuple)

- 튜플 요소 변경해보기

```
t = (1, 2, [1, 2])
t[1] = 3 #  오류 발생
t[2] = [1, 3]
del(t[1])

list1 = t[2]
list1[0] = 3
t[2][0] = 4
```

- 튜플의 구조를 바꾸는 것은 불가능

실습문제 3

- 문제
  - 정수 한 개(n)를 함수의 인자로 받고, 그 정수의 n개만큼 사용자로부터 정수를 입력 받고, 이를 요소로 튜플을 구성한 후 함수의 결과값으로 반환하는 함수 작성. 이 함수를 이용해서 5개의 정수 값으로 구성된 튜플을 생성하고 출력하는 프로그램 작성

- 요구사항
  - 사용자가 입력한 정수의 개수는 함수에 인자로 전달
  - 사용자로부터 입력 받는 값은 양의 정수만으로 가정
  - 함수를 검수할 때 입력 받아서 튜플로 만들 정수는 5개

실습문제 3

- 최종 코드

```
def createTupleFromInput(n):
    t = ()
    for i in range(n):
        num = int(input("정수를 입력하세요: "))
        t += (num,)
    return t

t1 = createTupleFromInput(5)
print(t1) # 튜플의 내용을 확인하기 위해 출력

for n in t1:
    print(n) # 각 요소 출력
```

실습문제 4

- 문제
  - 도형 정보를 담고 있는 튜플의 요소들을 이용해서 도형의 면적을 계산해서 출력하는 프로그램 작성
  - 도형 정보를 담고 있는 튜플의 예시
    - (“사각형”, 30, 20, “원”, 10)
- 요구사항
  - 튜플에 있는 도형의 개수는 정해져 있지 않음
  - 원주율은 math.pi 사용 (import math 필요)
  - calcAndPrintArea() 함수는 튜플을 인자로 받고 면적을 계산해서 화면에 출력
  - 튜플 예시: ("사각형", 30, 20, "원", 10, "사각형", 20, 40, "사각형", 10, 10, "원", 20) 사용
  - 출력 예시: 도형_종류, 면적계산시 필요정보, 넓이

실습문제 4

- 최종 코드

```
import math
def calcAndPrintArea(t):
    idx = 0
    while idx < len(t):
        if t[idx] == "사각형":
            area = t[idx + 1] * t[idx + 2]
            print(f"{t[idx]}, {t[idx + 1]}, {t[idx + 2]}, {area}")
            idx += 3
        elif t[idx] == "원":
            area = t[idx + 1] * math.pi
            print(f"{t[idx]}, {t[idx + 1]}, {area}")
            idx += 2
t = ("사각형", 30, 20, "원", 10, "사각형", 20, 40, "사각형", 10, 10, "원", 20)
calcAndPrintArea(t)
```

함수에서 여러 개 값을 반환하기

- 함수에서 여러 값을 함께 반환
  - 튜플이나 리스트를 사용해서 반환할 수 있음
  - 함수 결과를 받는 변수 개수는 1 또는 리스트/튜플의 길이

```
def funcReturnsTuple():
    return 1, 2

def funcReturnsList():
    return [3, 4, 5]

tuple1 = funcReturnsTuple()
list1 = funcReturnsList()
x, y = funcReturnsTuple()
a, b, c = funcReturnsList()
```

함수에서 여러 개 값을 반환하기

- 두 개의 숫자 값을 입력으로 전달 받고, 비교해서 작은 것, 큰 것 순서대로 값들을 다시 반환하는 함수를 작성
  - 숫자 비교
  - 작은 값, 큰 값 순서대로 튜플 또는 리스트로 구성
  - 튜플 또는 리스트 반환

함수에서 여러 개 값을 반환하기

- 튜플 버전

```
def orderNumbers(num1, num2):
    if num1 <= num2:
        return num1, num2
    else:
        return num2, num1
```

- 리스트 버전

```
def orderNumbers(num1, num2):
    if num1 <= num2:
        return [num1, num2]
    else:
        return [num2, num1]
```

리스트와 튜플 간 상호 변환

- 리스트를 튜플로 변환
  - tuple()함수 사용

```
list1 = [1, 2]
tuple1 = tuple(list1)
tuple2 = tuple([1, 2, 3])
```

- 튜플을 리스트로 변환
  - list()함수 사용

```
tuple1 = (1, 2)
list1 = list(tuple1)
list2 = list((1, 2, 3))
```