SYSC 1005 Introduction to Software Development

Loops and Iteration in Python

Yen-Chia Hsu

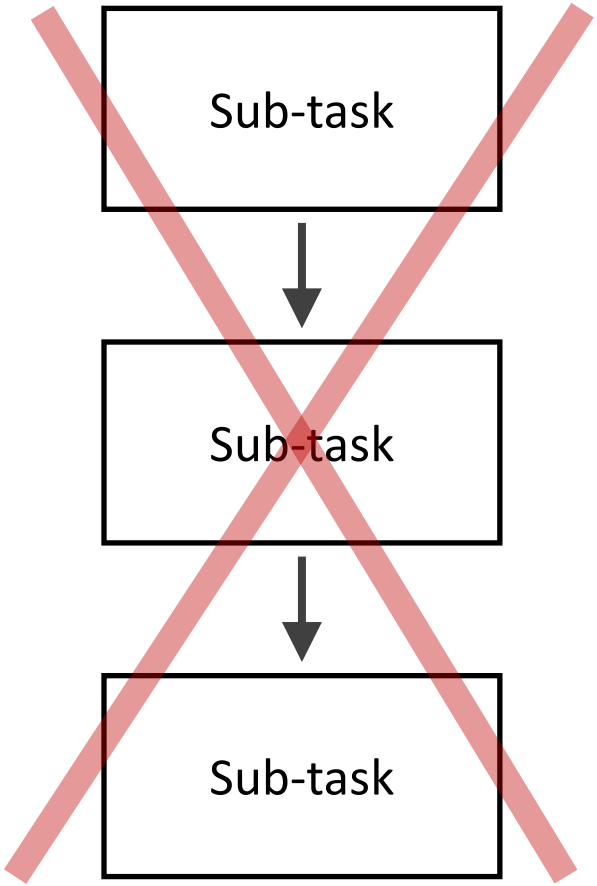
http://yenchiah.me

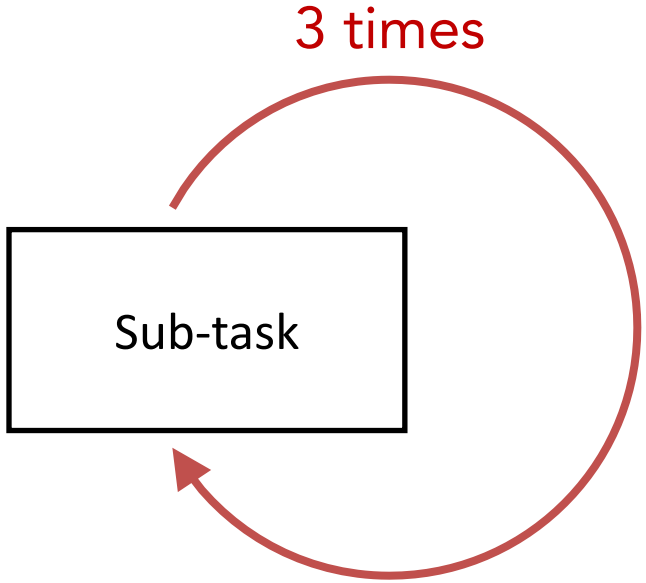
Project Scientist

Carnegie Mellon University
The Robotics Institute

CREATE Lab
Community Robotics, Education and Technology Empowerment

Loops offer a convenient way to do something over and over again (repeated sub-tasks).





Given an array of three numbers, multiply all numbers by 10, and return them in a new array using the original order.

Example:

- Input: [1, 2, 3]
- Output: [10, 20, 30]

One straightforward approach is to access each number in the input array, multiply it by 10, and put it into the output array.

```
a = [1,2,3]
b = []

num = a[0]*10
b.append(num)
print(b)

num = a[1]*10
b.append(num)
print(b)

num = a[2]*10
b.append(num)
print(b)
```

The input and the output array

Output is now [10]

Output is now [10, 20]

Output is now [10, 20, 30]

We can use a `while` loop statement to iteratively multiply each number by 10, which simplifies the program.

```
a = [1,2,3]
b = []

num = a[0]*10
b.append(num)
print(b)

num = a[1]*10
b.append(num)
print(b)

num = a[2]*10
b.append(num)
print(b)
```

Repeated sub-task

`len(a)` gives the length of an array

```
a = [1,2,3]
b = []
n = len(a)

i = 0
while i < n:
    num = a[i]*10
    b.append(num)
    print(b)
    i += 1
```

Another way is to use a `for` loop statement, which can stop the sub-task after a definite number of steps.

```
a = [1,2,3]
b = []

num = a[0]*10
b.append(num)
print(b)

num = a[1]*10
b.append(num)
print(b)

num = a[2]*10
b.append(num)
print(b)
```

Repeated sub-task

`len(a)` gives the length of an array

`range(n)` gives an array `[0,1,…,n-1]`

```
a = [1,2,3]
b = []
n = len(a)

for i in range(n):
    num = a[i]*10
    b.append(num)
    print(b)
```

Given an array of numbers with any size, multiply all numbers by 10, and return them using the original order.

Example:

- Input: [1, 2, 3, 4, 5, …]
- Output: [10, 20, 30, 40, 50, …]

Based on our previous program, we can re-write it into a function that can take the array input with arbitrary size.

```python
a = [1,2,3]
b = []
n = len(a)

for i in range(n):
    num = a[i]*10
    b.append(num)
    print(b)
```

Sub-task

```python
def process(a):
    b = []
    n = len(a)

    for i in range(n):
        num = a[i]*10
        b.append(num)
        print(b)

    return b


b = process([1,2,3])
```

- We can further make the loop statements more flexible to handle advanced tasks.

- What if we only want to process a part of the numbers, rather than all of them?

Given an array of numbers with any size, multiply all even numbers by 10, and return them using the original order.

Example:

- Input: [1, 2, 3, 4, 5, …]
- Output: [20, 40, …]

Based on our previous function, we can add the `if` and `continue` statement to control the flow in the loop.

```
def process(a):
    b = []
    n = len(a)

    for i in range(n):
        num = a[i]*10
        b.append(num)
        print(b)

    return b
```


```
def process(a):
    b = []
    n = len(a)

    for i in range(n):
        if a[i] % 2 == 1:
            continue
        num = a[i]*10
        b.append(num)
        print(b)

    return b
```

Given an array of numbers with any size, multiply only the first three numbers by 10, and return them using the original order.

Example:

- Input: [1, 2, 3, 4, 5, …]
- Output: [10, 20, 30]

Based on our previous function, we can add the `if` and `break` statement to control the flow in the loop.

```python
def process(a):
    b = []
    n = len(a)

    for i in range(n):
        num = a[i]*10
        b.append(num)
        print(b)

    return b
```


```python
def process(a):
    b = []
    n = len(a)

    for i in range(n):
        if i == 3:
            break
        num = a[i]*10
        b.append(num)
        print(b)

    return b
```

Q1:

It looks like that `for` and `while` can achieve that same thing. How can I choose between them?

A1:

It depends on your personal preference. In general, use `for` loop if you know the exact number of iterations, and use `while` loop otherwise.

Q2:

It looks like that `continue` and `break` are similar.

What are the differences between them?

A2:

Use the `continue` statement if you want to skip the current iteration based on some conditions. Use the `break` statement if you want to exit the entire loop.

Loops offer a convenient way to do something over and over again (repeated sub-tasks), such as processing data in arrays.

Table of Contents

4. More Control Flow Tools

Besides the `while` statement just introduced, Python uses the usual flow control statements known from other languages, with some twists.

4.1. `if` Statements

Perhaps the most well-known statement type is the `if` statement. For example:

```
>>> x = int(input("Please enter an integer: "))
Please enter an integer: 42
>>> if x < 0:
...     x = 0
...     print('Negative changed to zero')
... elif x == 0:
...     print('Zero')
... elif x == 1:
...     print('Single')
... else:
...     print('More')
...
More
```

More information can be found at https://docs.python.org/3/tutorial/controlflow.html

SYSC 1005 Introduction to Software Development

Teaching Approach

- Use examples to guide students so that they can connect real-world applications and the abstraction of programming.

- Start from simple cases, and then gradually increase the difficulty of building confidence in problem-solving.

- Besides teaching the language syntax, also explain the significance (why the syntax is useful).

- **Flipped Classroom:**

  In the future, besides pre-recorded lectures, I will also use real-time Q&A sessions, online discussion forum, and online Q&A documents to increase engagement.

- **Learning by Doing:**

  In the future, I will give students code templates before class and ask them to implement the code together, by pausing the lecture video at specific steps.