



Loops



Outline

- Updating variables
- While loop
 - Updating variables
 - Break & continue
 - Infinite loops
- For loop
 - Iterable objects - string, list, range, (dictionary, tuple, set)
 - Updating variables
 - Break & continue
 - Applications - adding up, counter, maximum number
 - Nested loops

Loop statements

- Computers are often used to automate repetitive tasks.
- A loop is a control structure that allows you to execute a block of code repeatedly based on a specific condition or for a fixed number of times.
 - While loop
 - For loop

Updating variables

- A common pattern in loops statements is an assignment statement that **updates a variable**, where the new value of the variable depends on the old.

```
x = 10  
  
x = x + 1  
print (x) } x = 10 + 1 = 11  
  
x = x + 1  
print (x) } x = 11 + 1 = 12  
  
x = x + 1  
print (x) } x = 12 + 1 = 13
```

11
12
13

```
x = 10  
  
x = x - 1  
print (x) } x = 10 - 1 = 9  
  
x = x - 1  
print (x) } x = 9 - 1 = 8  
  
x = x - 1  
print (x) } x = 8 - 1 = 7
```

9
8
7

Updating variables

- Assignment operators are used to assign values to variables.

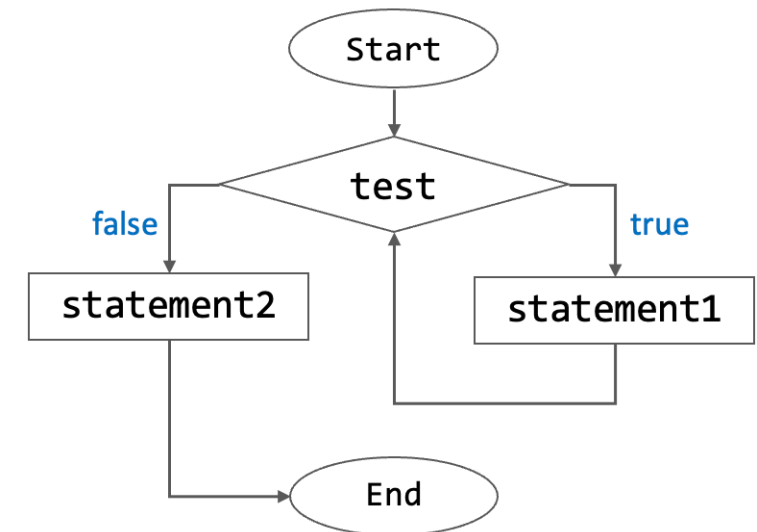
Assignment operator	Example	Same as
=	$x = 1$	$x = 1$
+=	$x += 1$	$x = x + 1$
-=	$x -= 1$	$x = x - 1$
*=	$x *= 2$	$x = x * 2$
/=	$x /= 2$	$x = x / 2$

While loop

- A while statement repeatedly executes a block of statements as long as a test (condition) at the top keeps evaluating to a **true** value.
- It is called a "loop" because it repeatedly returns control to the beginning of the loop statement until the specified condition becomes false.

General format

```
While <test>:           # loop test  
    <statement1>        # loop body  
else:                  # optional else  
    <statement2>        # run if the test is false
```



While loop

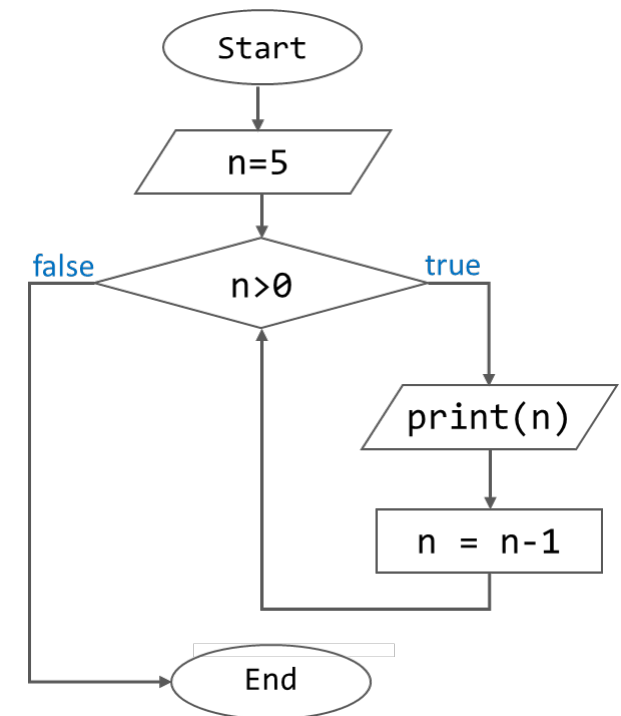
- Example:

- While n is greater than 0, print the value of n and then decrement the value of n by 1.

```
n = 5
while n > 0:    → # loop test
    print(n)    } # loop body
    n = n - 1
```

5
4
3
2
1

Iteration	n	n>0	n - 1
1	5	True	5 - 1 = 4
2	4	True	4 - 1 = 3
3	3	True	3 - 1 = 2
4	2	True	2 - 1 = 1
5	1	True	1 - 1 = 0
6	0	False	

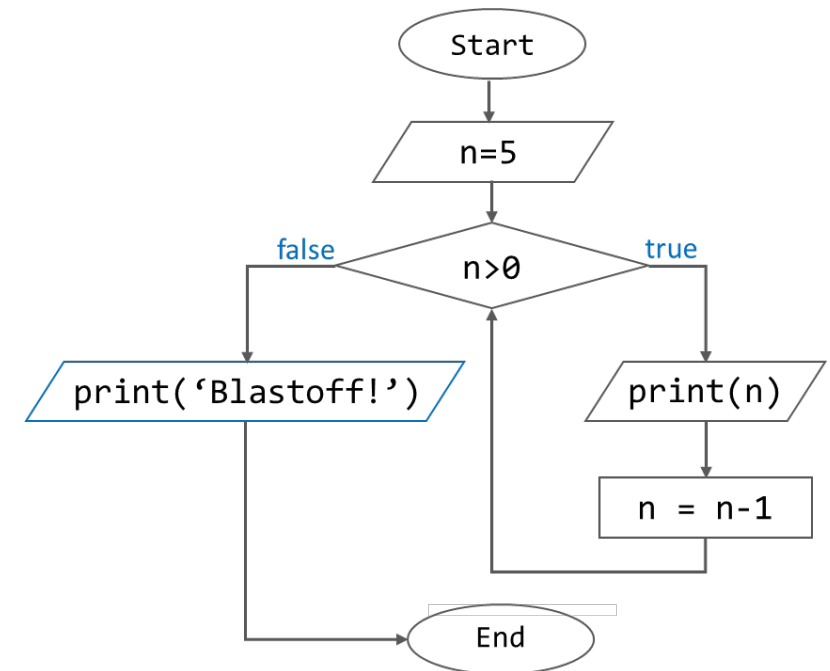


While loop - else

- The statements inside the `else` clause are executed only when the test is false.

```
n = 5
while n > 0:
    print(n)
    n = n - 1
else:
    print("Blastoff!") # run this line if the test is false
```

```
5
4
3
2
1
Blastoff!
```



Exercise

(A.1) Define a variable `n=1` . Write a program to print out the following results.

```
1
2
3
4
5
```

(A.2) Define a variable `n=2` . Write a program to print out the following results.

```
2
4
6
8
10
done
```

While loop - break and continue

- Use `break` and `continue` statements within the body of a while (or for) loop.

General format

```
While <test1>:  
    <statement1>  
    if <test2>:  
        break          # Terminate the loop immediately, skipping "else"  
    if <test3>:  
        continue       # Go to the top of the loop (test1), skipping "statement 2".  
    <statement2>  
else:  
    <statement3>        # When the loop condition is False without hitting a 'break'
```

While loop - break

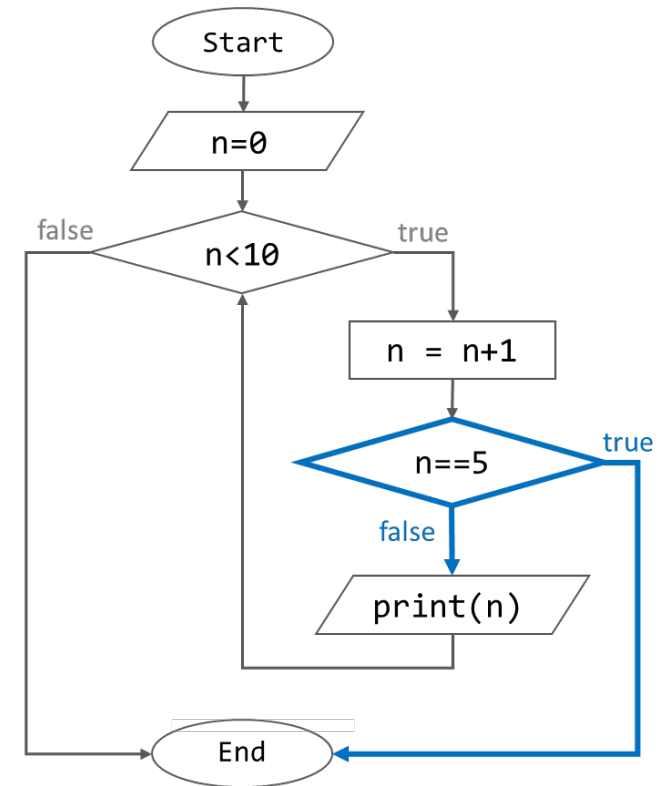
- The break statement causes an immediate **exit from a loop**.

```
n = 0
while n < 10:
    n = n + 1

    if n == 5:
        break

    print(n)
```

1
2
3
4



While loop - continue

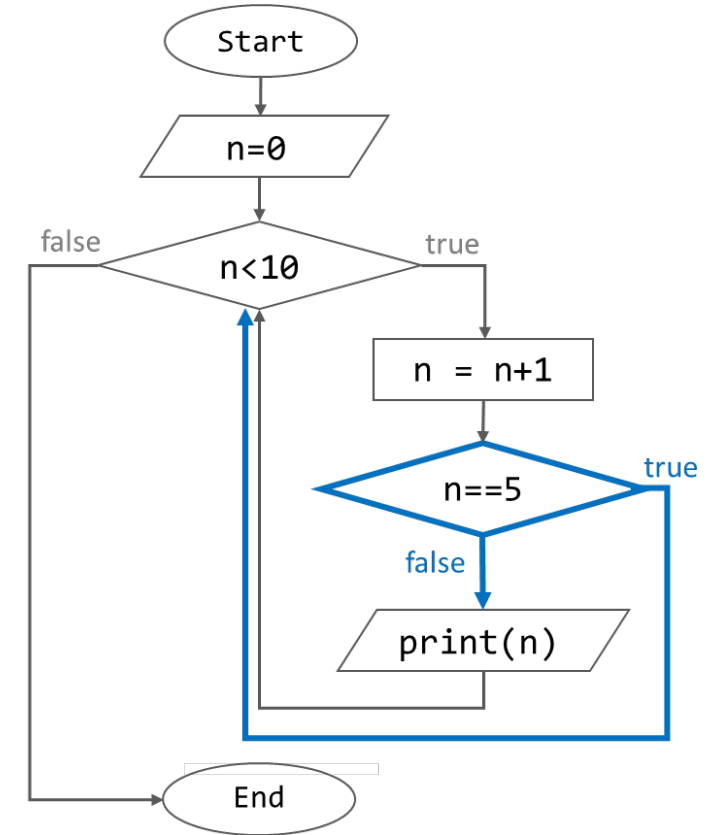
- The continue statement causes an immediate **jump to the top of a loop**.

```
n = 0
while n < 10:
    n = n + 1

    if n == 5:
        continue

    print(n)
```

1
2
3
4
6
7
8
9
10




Infinite loops

- In a while loop, when the condition always evaluates to true, you create an indefinite loop.

```
In [*]: n = 10
while True:
    print(n)
    n = n - 1
```

9
8
7
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10



- Interrupt the kernel.
- Rewrite your code (e.g., add a break statement).

Exercise

Exercise.B

(B.1) Assume you have a part-time job with a daily income of 300 kr, and your goal is to reach a total of 2500 kr. You work for two days, then have a day off, and continue this pattern until you reach or exceed your target income of 2500 kr. Write a program using a while loop to calculate and print out the cumulative income for each working day until you reach your goal.

Expected output:

Day-1: 300

Day-2: 600

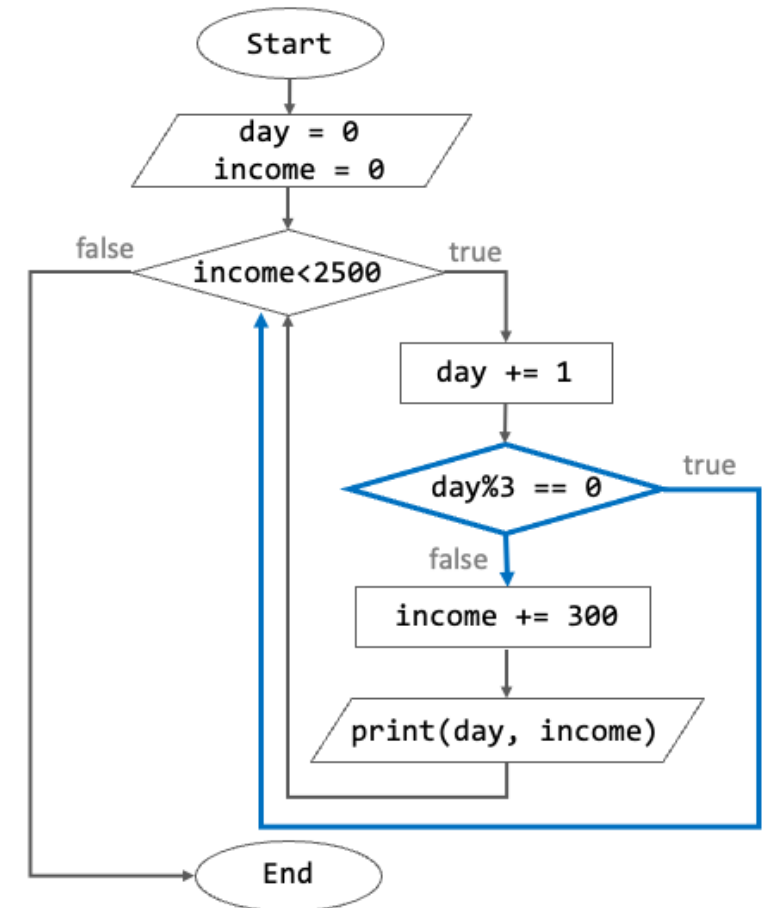
Day-4: 900

Day-5: 1200

...

Day-13: 2700

Day-1	Day-2	Day-3	Day-4	Day-5	Day-6	...
+300	+300	X	+300	+300	X	...
Cumulative income	300	600	900	1200	1200	...



For loop

WHILE loops & FOR loops

- Using `while` statement is used to create an **indefinite loop**
 - An indefinite loop continues to run until a specified condition becomes False.
- Using `for` statement is used to create a **definite loop**
 - A definite loop iterates through a known set of items, running through as many iterations as there are items in the set .

For loop

- A **for** loop can iterate through an **iterable object**.
 - string, list, range, dictionary, tuple, set

General format

```
for <target> in <object>:    # Assign the next item in the object to the target
    <statements>            # Repeat loop body use target
```

```
mystr = "python"
for i in mystr:
    print(i)
```

p
y
t
h
o
n

i → target (loop variable)
mystr → object
print(i) → statement

Iteration	i	print (i)
1	i = "p"	p
2	i = "y"	y
3	i = "t"	t
4	i = "h"	h
5	i = "o"	o
6	i = "n"	n

For loop - list

- In each iteration, assign an item from the list to the loop variable **i**.

```
mylist = [5, 4, 3, 2, 1]
for i in mylist:
    print(i)
```

5
4
3
2
1

```
mylist = ['John', 'Leo', 'Emma']
for i in mylist:
    print('Happy New Year,', i)
```

Happy New Year, John
Happy New Year, Leo
Happy New Year, Emma

Iteration	i	print (i)
1	i = 5	5
2	i = 4	4
3	i = 3	3
4	i = 2	2
5	i = 1	1

Iteration	i	print ('Happy New Year", i)
1	i = 'John'	Happy New Year, John
2	i = 'Leo'	Happy New Year, Leo
3	i = 'Emma'	Happy New Year, Emma

For loop - range

- The `range()` function is used to generate a sequence of numbers, starting from `0` by default, and `incrementing by 1` (by default), and stopping before the specified number.

```
for i in range(10): #same as range(0,10)
    print(i)
```

0
1
2
3
4
5
6
7
8
9

```
for i in range(1,5):
    print(i)
```

Not including 5

1
2
3
4

```
for i in range(5):
    print("hello world")
```

hello world
hello world
hello world
hello world
hello world

- `range(stop)`: Generates a sequence of numbers from `0` to `stop-1`.
- `range(start, stop)`: Generates a sequence of numbers from `start` to `stop-1`.
- `range(start, stop, step)`: Generates a sequence of numbers from `start` to `stop-1`, incrementing by `step` in each iteration.

Exercise

Exercise.C

(C.1) Given the list below, use a for loop to generate the following output for each item in the list.

Expected result:

Apple has been added to your shopping cart.

Yogurt has been added to your shopping cart.

Avocado has been added to your shopping cart.

Salmon has been added to your shopping cart.

```
item_list = ["Apple", "Yogurt", "Avocado", "Salmon"]
```

(C.2) Print a sequence of numbers from 15 to 25 on one line.

Expected result:

15 16 17 18 19 20 21 22 23 24 25

For loop - updating variables

- A common pattern in loops statements is to **update a variable**, where the new value of the variable depends on the old value.

```
# Initialise a variable  
x = 0  
  
# For each iteration, we update the variable "x"  
for i in range(0,5):  
    x = x + i  
    print(x)
```

0
1
3
6
10

Iteration	i	x + i
1	0	0 + 0 = 0
2	1	0 + 1 = 1
3	2	1 + 2 = 3
4	3	3 + 3 = 6
5	4	6 + 4 = 10

For loop - updating variables

- Example: Calculate the cumulative number of cases per day

```
# Each item represents the number of confirmed cases per day
covid_list = [10, 15, 12, 10, 18]

# Initialise a variable (cumulative number of cases)
cum_case = 0

# For each iteration, we update the variable "cum_case"
for daily_case in covid_list:
    cum_case = cum_case + daily_case
    print(cum_case)
```

```
10      Day1
25      Day1 + Day2
37      Day1 + Day2 + Day3
47      Day1 + Day2 + Day3 + Day4
65      Day1 + Day2 + Day3 + Day4 + Day5
```

Iteration	daily_case	cum_case
1	10	0 + 10 = 10
2	15	10 + 15 = 25
3	12	25 + 12 = 37
4	10	37 + 10 = 47
5	18	47 + 18 = 65

Exercise

Exercise.D

(D.1) You have taken five consecutive daily quizzes, and you want to calculate your cumulative score for each day. Given the following list of quiz scores for each day, use a for loop to print out the cumulative score of each day.

Expected result:

15
25
38
55
67

```
scores = [15, 10, 13, 17, 12]
```

For loop - break & continue

- Use `break` and `continue` in a for loop body.

```
for i in range(0,10):  
    if i == 5:  
        break # Exit the loop now  
    print(i)
```

0
1
2
3
4

```
for i in range(0,10):  
    if i == 5:  
        continue # Go to the top of the loop  
    print(i)
```

0
1
2
3
4
6
7
8
9

For loop - Application (1) adding up

- Updating a variable in the for loop body.

```
# Initialise a variable  
total = 0  
  
# For each iteration, we update the variable "total"  
for n in range(1,11):  
    total = total + n  
    print(total)
```

```
1  
3  
6  
10  
15  
21  
28  
36  
45  
55
```

For loop - Application (2) counter

- Use a variable as a **counter** in a for loop's body.

```
animal_list = ['dog', 'dog', 'cat', 'dog', 'cat', 'cat', 'dog', 'cat', 'dog', 'cat']

# Initialise a variable
counter = 0

# For each iteration, we update the variable "counter"
for animal in animal_list:
    if animal == 'dog':
        counter = counter + 1

print('# of dogs:', counter)
```

```
# of dogs: 5
```

For loop - Application (3) find maximum number

- Find the maximum number in a list by iterating over the list and keeping track of the maximum value encountered so far.

```
price_list = [100, 450, 200, 250, 300, 500, 150]

# Initialise a variable
max_price = price_list[0] → Assume the first element is the maximum.

# For each iteration, if the test is true, we update the variable "max_price"
for price in price_list:
    if price > max_price:
        max_price = price

print("Maximum:", max_price)
```

Maximum: 500

Iteration	price	max_price
1	100	100
2	450	450
3	200	450
4	250	450
5	300	450
6	500	500
7	150	500

Exercise

Exercise.E

(E.1) Print all numbers from the list `numbers` until you encounter a number that is divisible by 7.

```
numbers = [88, 36, 139, 12, 24, 158, 29, 16, 152, 98, 45, 184, 191, 92, 117]
```

(E.2) Given the following list. How many names start with "S"?

Hint: Initialize a variable as a counter.

```
name_list = ["Henry", "Victoria", "Isaac", "Sara", "Maya", "Zoe", "Isabelle", "Nora", "Madelyn",  
            "Sophia", "Charlotte", "Michael", "Sebastian", "Leah", "Ryan", 'Matthew', "Mila"]
```

For loop - nested loop

- A nested loop is a loop inside a loop.
- For example, nested loops can be used to iterate over the elements of two or more lists to generate combinations of elements.

```
list_1 = ["A", "B", "C"]
list_2 = [1, 2, 3, 4]

for i in list_1:
    for j in list_2:
        print(i, j)
```

Outer loop { Inner loop }

A 1
A 2
A 3
A 4
B 1
B 2
B 3
B 4
C 1
C 2
C 3
C 4

i	j	print(i, j)
A	1	A 1
	2	A 2
	3	A 3
	4	A 4
B	1	B 1
	2	B 2
	3	B 3
	4	B 4
C	1	C 1
	2	C 2
	3	C 3
	4	C 4

For loop - nested loop

- Example:

- In each iteration of outer for loop, the inner for loop execute 3 times to print the current i and j.

```
for i in range(1,4):  
    for j in range(1,4):  
        print (f"{i}-{j}")
```

```
1-1  
1-2  
1-3  
2-1  
2-2  
2-3  
3-1  
3-2  
3-3
```

```
for i in range(1,4):  
    for j in range(1,4):  
        print (f"{i}-{j}", end = " ")  
    print()
```

```
1-1 1-2 1-3  
2-1 2-2 2-3  
3-1 3-2 3-3
```

Start a new line after each iteration of the outer for loop.

Exercise

Exercise.F

(F.1) Suppose you want to order a drink and a dessert, print out all possible combinations.

Example:

Tea & Carrot cake

Tea & Lemon tart

...

...

Juice & Cheesecake

```
menu_drink = ["Tea", "Coffee", "Juice"]
menu_dessert = ["Carrot cake", "Lemon tart", "Tiramisu", "Cheesecake"]
```

(F.2) Write a program to print out the following results.

1+1=2

1+2=3

1+3=4

2+1=3

2+2=4

2+3=5

3+1=4

3+2=5

3+3=6

For loop - Enumerate

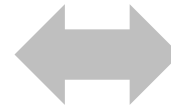


- In Python, you can iterate over the elements of a list without using the indexes.

```
item_list = ["A", "B", "C"]
```

```
for i in range(len(item_list)):
    print(item_list[i])
```

A
B
C



```
for item in item_list:
    print(item)
```

A
B
C

more commonly used in Python

- To track both the index and the element, you can use `enumerate()`.

```
for i in range(len(item_list)):
    print(f"index-{i}: {item_list[i]}")
```

index-0: A
index-1: B
index-2: C



```
for i, item in enumerate(item_list):
    print(f"index-{i}: {item}")
```

index-0: A
index-1: B
index-2: C

more commonly used in Python