



Python Basics, Conditionals, Loops & Functions

Lecture Flow

- Python Basics
- Conditionals
- Loops
- Functions



Python Basics



Why Python?

Python - Why?

- It is easier to implement **complex** algorithms and data structures.
- To make sure everyone has **equal** understanding of programming.
- We use Python for **lectures**.

Syntax

- No semicolons, yay?
- White Spaces matter.
- Similar to the English language.



// Java

```
public class HelloWorldApp {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

Python

```
print('Hello, world!')
```

Syntax - Indentation

- In Python, unlike other programming languages, **indentation** serves a crucial purpose beyond just **readability**.
- Python uses **indentation** as a way to indicate and define **blocks of code**.

Variables

- Variables are used to **store** and **manipulate data**.
- Python has no command for **declaring** a **variable**.
- They are **created** by assigning a **value** to a **name**.
- Python has **dynamic typing**.

```
x = 4          # x is of type int  
x = "A2SV"     # x is now of type str
```

Variables- Names

- Must **start** with a **letter** or the **underscore** character
- Can **not start** with a **number**
- Can **only contain** **alphanumeric** characters and **underscores** (A-z, 0-9, and _)
- Case-sensitive (age, Age and AGE are three different variables)
- Can **not be** a **keyword** (if, while, and, for, ...)
- snake_case

Data Types in Python

- Data types define the **kind of data** that can be **stored** and **manipulated** in a program.
- Common Built-in Data Types:
 - Boolean (bool)
 - Integer (int)
 - Float(float)
 - String (str)

Boolean

- In programming you often need to know if an expression is **True** or **False**.
- You can evaluate any expression in Python, and get one of two answers, **True** or **False**.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer:

```
10 > 9 # True
10 == 9 # False
10 < 9 # False
```

Boolean- Evaluation

- The **bool()** function allows you to evaluate any value, and give you **True** or **False** in return,
- In Python values with content are True:
 - Any string is **True** , except empty strings.
 - Any number is **True** , except **0**.
 - Any list, tuple, set, and dictionary are **True** , except empty ones.

Numeric data types

- Integer:
 - Represent integer numbers without any decimal points
 - Can be positive or negative numbers, or zero.
 - Examples of integers are: $x = -5$, $x = 0$, $x = 10$, $x = 100$

Numeric data types

- Float:
 - Represent decimal numbers or numbers with a fractional part
 - They are written with a decimal point, even if the fractional part is zero
 - Examples of floating-point numbers are: **x = -2.5**, **x = 3.14**, **x = 1.0**

Operators

- Operators are used to perform operations on variables and values.
- In the example below, we use the + operator to add together two values:

```
print(10 + 5)
```


Operators

- Python divides the operators in the following groups:
 - Arithmetic operators
 - Assignment operators
 - Comparison operators
 - Logical operators
 - Identity operators
 - Membership operators

Operators- Arithmetic

- Arithmetic operators are used with **numeric values** to perform common **mathematical operations**.

Operators- Arithmetic

Operator	Name	Example
+	Addition	$x + y$
-	Subtraction	$x - y$
*	Multiplication	$x * y$
/	Division	x / y
%	Modulus	$x \% y$
**	Exponentiation	$x ** y$
//	Floor division	$x // y$

Operators- Precedence

Operator	Description
<code>()</code>	Parentheses
<code>**</code>	Exponentiation
<code>+x</code> <code>-x</code> <code>~x</code>	Unary plus, unary minus, and bitwise NOT
<code>*</code> <code>/</code> <code>//</code> <code>%</code>	Multiplication, division, floor division, and modulus
<code>+</code> <code>-</code>	Addition and subtraction

Operators- Comparison

- Comparison Operators are used to compare two values.

Operators- Comparison

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Practice Problems

- [Arithmetic Operators](#)
- [Division](#)
- [Convert the Temperature](#)
- [Palindrome Number](#)

Operators- Logical

- Logical operators are used to combine conditional statements.

Operators- Logical

Operator	Description	Example
and	Returns True if both statements are true	<code>x < 5 and x < 10</code>
or	Returns True if one of the statements is true	<code>x < 5 or x < 4</code>
not	Reverse the result, returns False if the result is true	<code>not(x < 5 and x < 10)</code>

Operators- Identity

- Identity Operators are used to **compare** the objects, not if they are equal, but if they are actually the **same object**, with the **same memory location**.

Operators- Identity

Operator	Description	Example
is	Returns True if both variables are the same object	x is y
is not	Returns True if both variables are not the same object	x is not y

Operators- Membership

- Membership Operators are used to **test** if **a sequence** is **presented** in an **object**.

Operators- Membership

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Check Point

Strings

- Strings in python are surrounded by either single quotation marks, or double quotation marks.
- **'hello'** is the same as **"hello"**.
- You can display a string literal with the **print()** function:
- String in python are **immutable**.
- You can assign a multiline string to a variable by using three quotes:

```
a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""
```

Strings - Slicing Strings

- You can **return** a range of **characters** by using the **slice syntax**.
- **Specify** the **start index** and the **end index**, separated by a colon, to return a **part** of the string. We can also specify **step** as a third parameter (optional).

```
b = "Hello, World"
print(b[2])
print(b[-3])
print(b[2:5])
print(b[:5])
print(b[2:])
print(b[5:2:-1])
print(b[::-1])
print(b[::-2])
```


Strings - String Concatenation

- To concatenate, or combine, two strings you can use the + operator.

```
a = "Hello"  
b = "World"  
c = a + b  
print(c)
```

Strings - Formatting

- To **format** strings in python we can use **f-strings**.

```
a = 1
b = "hello"
print(f"{b} {a} {a + 2}") # hello 1 3
```

Strings - Substring search

- In python we can use the "in" operator to check if a string occurs as a substring of another string

```
print("Hello" in "Hello world")
```

Variables - Casting

- Variable casting allows **converting** a **value** from one **data type** to **another**.
- Python provides built-in functions for explicit casting, such as **'str()'**, **'int()'**, and **'float()'**.

```
y = int(3.0) # y will be 3
z = float(3) # z will be 3.0
```

Check point

- What will be the output of the following statements?

```
s = "Hello, World!"
```

```
print(s[5]) # ?
```

```
print(s[-2]) # ?
```

```
print(s[1:]) # ?
```

```
print(s[-2:]) # ?
```

Practice Problems

- [sWAP cASE](#)
- [String Split and Join](#)
- [What's Your Name?](#)

Conditionals

If statement

- We use **if** statement to write a single alternative decision structure.
- Here is the general format of the if statement:

if condition:
 statement
 statement

```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```


Elif

- The **elif** keyword is python's way of saying "if the previous conditions were not true, then try this condition".

```
a = 33
b = 33
if b > 33:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

Else

- The **else** keyword catches anything which isn't caught by the preceding conditions.

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

Conditionals

- You can have **if** statements inside **if** statements, this is called nested **if** statements.
- We can use logical operators to combine conditional statements.

```
a = 200
b = 33
c = 500
if a > b and c > a:
    print("Both conditions are True")
```

Loops

While Loop

- With the **while** loop we can execute a set of statements as long as a condition is true.

```
i = 1
while i < 6:
    print(i)
    i += 1
```

For Loop

- A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

```
fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
    print(x)
```

Nested Loops

- A nested loop is a loop inside a loop.
- The "inner loop" will be executed one time for each iteration of the "outer loop":

```
adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]
for x in adj:
    for y in fruits:
        print(x, y)
```

The range() Function

- The **range()** function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.

```
for x in range(6): # init(d) 0 final 6 step(d) 1
    print(x)
```

```
for x in range(2, 6): # init 2 final 6 step(d) 1
    print(x)
```

```
for x in range(2, 10, 2): # init 2 final 10 step 2
    print(x)
```


Continue Statement

- With the **continue** statement we can stop the current iteration, and continue with the next.

```
i = 0
while i < 9:
    i += 1
    if i == 3:
        continue
    print(i)
```

```
for i in range(9):
    if i == 3:
        continue
    print(i)
```

Break Statement

- With the **break** statement we can stop the loop even if the while condition is true:

```
for i in range(9):  
    if i > 3:  
        break  
    print(i)
```

```
i = 1  
while i < 9:  
    print(i)  
    if i == 3:  
        break  
    i += 1
```

For - Else

- With the **else** statement we can check if the loop finishes its iteration or not.

```
fruits = ["apple", "banana", "mango",  
"orange"]  
  
for fruit in fruits:  
    if fruit == "pineapple":  
        print("Found pineapple!")  
        break  
else:  
    print("Pineapple not found in the  
list.")
```

Check point

- What is the output of the following nested Loop?

```
for num in range(10,14):  
    for i in range(2, num):  
        if num % i == 1:  
            print(num)  
            break
```

A) 10
11
12
13

B) 11
13

Functions

Functions

- A function is a reusable **block of code** which only runs when it is **called**.
- You can **pass** data, known as **parameters**, into a **function**.
- A function can **return data** as a **result**.

```
def my_function():  
    print("Hello from a function")  
my_function()
```

Arguments

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses.
- You can add as many arguments as you want, just separate them with a comma.

```
def my_function(fname):  
    fname[0] = "anna"  
    print(fname[0] + " Refsnes")  
data = ["Emil"]  
my_function(data)  
print(data[0])
```

Return Values

- To let a function **return** a value, use the return statement:

```
def my_function(x):  
    return 5 * x  
  
print(my_function(3))  
print(my_function(5))  
print(my_function(9))
```


Lambda

- A lambda function is a small **anonymous function**.
- A lambda function can take any number of arguments, but can only have one expression.
- Syntax:

lambda arguments : expression

```
x = lambda a : a + 10  
print(x(5))
```

```
x = lambda a, b : a * b  
print(x(5, 6))
```

Practice Problems

- [Smallest even multiple](#)
- [Weird](#)
- [Powers](#)
- [Mod Power](#)
- [Longest Common Prefix](#)
- [More exercise](#)

Quote of the day

“A year from now you may wish you had started today”

– Karen Lamb