

Programming in Python

While Loops

Repeated Code

- Let's say we want the user to enter 3 scores.
- Then our program will calculate the average and print it.

```
total = 0
score1 = int(input("Enter score #1: "))
total = total + score1
score2 = int(input("Enter score #2: "))
total += score2
score3 = int(input("Enter score #3: "))
total += score3
average = total / 3
print("Average =", average)
```

```
# equivalent to
total = total + score1
```

```
Enter score #1: 97
Enter score #2: 82
Enter score #3: 91
Average = 90.0
```

Repeated Code

- Now we want to average 30 scores.

```
total = 0
score1 = int(input("Enter score #1: "))
total += score1
score2 = int(input("Enter score #2: "))
total += score2

# copy the two lines 27 more times,
# changing the number on each line

score30 = int(input("Enter score #30: "))
total += score30
average = total / 30
print("Average =", average)
```

Loops

- Let's use loops!
- There are 2 types of loops in Python
 - › `while`
 - › `for`



while Loop

- A **while** loop is similar to an **if** statement.
- As long as the condition (Boolean expression) is **True** the block (loop body) is executed.
- When the condition is **False**, the loop exits and the program continues.
- Pseudocode



Indentation
matters!

```
while condition:  
    # condition is True  
    statement  
    # there may be more statements  
    # now the condition is False
```

while Loop

- Let's average 30 scores.

```
total = 0
num = 1

while num <= 30:
    score = int(input("Enter score: "))
    total += score
    num += 1

average = total / 30
print("Average =", average)
```

```
Enter score: 97
Enter score: 82
# 27 more times
Enter score: 91
Average = 90.0
```

```
# equivalent to
total = total + score
num = num + 1
```

Structure of while Loop

```
# create new variable  
while condition:  
    # test variable in condition  True  
    # update variable  
    # other statements  
# now the condition is False
```



Consider

- What will happen?

```
counter = 10
while counter > 0:
    print(counter)
    counter = counter - 1
print("Blast off!")
```

```
counter = 10
while counter > 0:
    print(counter)
    counter = counter + 1
print("Blast off!")
```

```
10
9
8
7
6
5
4
3
2
1
Blast off!
```

```
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

Infinite Loop

- A loop that always repeats without ending is an infinite loop.
- This happens when the Boolean expression never becomes **False**.
- There should always be a way to exit a loop.
- The best practice is to update the variable that you are testing in the Boolean expression.



Loops

- Let's execute the loop at least once.

```
keepGoing = 1
while keepGoing != 0:
    print("The loop keeps running!")
    keepGoing = int(input("Enter 0 to quit: "))
```

```
The loop keeps running!
Enter 0 to quit: 5
The loop keeps running!
Enter 0 to quit: 24
The loop keeps running!
Enter 0 to quit: 0
```

Loops

- Let's execute the loop at least once using a string.

```
name = "unknown"
while name != "Pikachu":
    print("I'm looking for Pikachu!")
    name = input("Which Pokemon are you? ")
print("I found Pikachu!")
```

```
I'm looking for Pikachu!
Which Pokemon are you? Charmander
I'm looking for Pikachu!
Which Pokemon are you? Pikachu
I found Pikachu!
```

Programming in Python

Review till Week 4

Fun with Printing

- `print()` is a function that has already been created for us.
- When you write `print("Hello World!")`, you are calling the `print()` function with the input of a string containing "Hello World!".
- By default, the input string will be printed on a new line.

Code

```
print("First line")
print("Second line")
```

```
print(First line)
```

Console

```
First line
Second line
```

```
Error
```

Print Strings

- Text-based data in Python is handled with **str** objects, or strings.
- Strings in Python are written in a variety of ways:
 - › Single quotes: 'Hello World!'
 - › Double quotes: "Hello World!"
 - › Triple single quotes: """Hello World!"""
 - › Triple double quotes: """Hello World!"""
- A string in Python is immutable.
 - › That means it will never change its value.
 - › If you modify a string, you are actually creating a new string.



Escape Characters

- An escape character:
 - › Deviates (escapes from) the normal meaning
 - › Is indicated by 2 characters (backslash and another character)
 - › Allows print to display something special
- Here are some of the more popular escape characters:

Escape Character	What's Printed
\n	new line
\t	tab
\'	single quote
\"	double quote
\\	one backslash

Solution

- If using double quotes to delimit the string:

```
print("What does the word \"kumak\" mean?")
```

- If using single quotes to delimit the string, then we don't have to use escape characters:

```
print('What does the word "kumak" mean?')
```

- Result either way:

What does the word "kumak" mean?

Comments

- Comments are skipped by Python
- Intention is to provide reader (or maintainer) extra information to understand the code

```
# This is a single line comment
```

```
"""
This is a
multi line
comment
"""
```

Variables

- Syntax (pseudocode)

```
variable = value
```

```
variable = expression
```

- Code

```
age = 20
```

```
total = 5 + 3
```

- The **=** (equals sign) means assignment.
 - › Take the number **20** and store it in a variable named **age**.
- The variable has to be on the left side of the **=**.
 - › The computer will evaluate the expression on the right side.
 - › Then the variable will be able to access that information.

Valid Variable Names

- Start with a letter (a-z, A-Z) or underscore (_)
- Can contain digits (0-9)
- No spaces or other characters
- Two conventions:
 - › camelCase
 - Used in lots of languages
 - Examples: numStudents, isRainingNow
 - › snake_case
 - More Pythonic
 - Examples: num_students, is_raining



Primitive Variable Types

- Python has various types of variables
 - › Primitive types (we will learn today)
 - › Non-primitive types (we will learn in a future level)
- The primitive data types in Python are:

Data Type	Description	Examples
<code>str</code>	a string of characters	<code>greet = "Hello World!"</code>
<code>int</code>	an integer or whole number that can be positive or negative	<code>num = 42</code> <code>zero = 0</code> <code>negative = -1</code>
<code>float</code>	a floating point or decimal number	<code>pi = 3.1415</code> <code>neg = -12.5</code>
<code>bool</code>	a boolean has a value of True or False	<code>isLearning = True</code> <code>isSnowing = False</code>

Variables

- Variables have 3 parts
 - › name (or label)
 - › type (such as str, int, float, bool)
 - › value (determines the type)

- Create a variable

```
variable = value
```

- Examples

```
greeting = "Good day"
course = "2: Python"
year = 2023
half = 0.5
isCool = True
```

Print function

```
print(object(s) , sep=separator , end=end)
```

Parameter	Description
object(s)	Any object, and as many as you like. Will be converted to string before printed
sep	Optional. Specify how to separate the objects, if there is more than one. Default is '' (one space)
end	Optional. Specify what to print at the end. Default is '\n' (new line)

- When we call the `print()` function, anything we put in the parenthesis () is input.
- There is no output.

String Concatenation

- Use the plus (+) to add two strings together is string concatenation.
- We are concatenating two strings together.
- The result is a string (str).

```
code = "TECH"  
num = "2"  
print(code + num)  
course = code + "-" + num  
print(course)
```

TECH2
TECH-2

```
code = "TECH"  
num = 2  
print(code + num)
```

ERROR

Convert to String

- You can use the `str()` function on lots of types of variables including
 - › `int`
 - › `float`
 - › `bool`

```
weather = "It is snowing"
isSnowing = False
print("\\" + weather + "\" is " + str(isSnowing))
```

```
"It is snowing" is False
```

Getting User Input

- To get input from the user, use the `input()` function.
 - › When we call `input()`, we put a message to the user as a string.

```
var = input("This is a message to the user")
```

- The function will give us a **string** with their response from the console window.
 - › We need to have a variable waiting to capture the information.

```
name = input("Enter your name: ")  
print("Hi, " + name)
```

```
Enter your name: Rashi  
Hi, Rashi
```

Text in green
is user input

Reading in Numbers

- What will happen?

```
num1 = input("Enter a number: ")  
num2 = input("Enter another number: ")  
print(num1 + num2)
```

```
Enter a number: 8  
Enter another number: 4  
84
```

Reading in Numbers

- What happened?
 - › The `input()` function always returns a string.
 - › The `+` combines (or concatenates) two strings together.
- Solution:
 - › When you want numbers, convert them to an `int` or `float`.

Function	Description	Example	Returns
<code>float(x)</code>	Returns a floating-point value by converting x	<code>float("10.0")</code>	<code>10.0</code>
<code>int(x)</code>	Returns an integer value by converting x	<code>int("10")</code>	<code>10</code>
<code>str(x)</code>	Returns a string value by converting x	<code>str(10)</code>	<code>"10"</code>

Reading in Numbers

- Updated code:

```
num1 = int(input("Enter a number: "))  
num2 = int(input("Enter another number: "))  
print(num1 + num2)
```

```
Enter a number: 8  
Enter another number: 4  
12
```

Arithmetic Operators

- To store numbers, Python has two types of variables: `int` and `float`.
- We can do mathematical operations with each.
- Arithmetic operators that are similar to a calculator:

+ - × ÷

+ - * /

- Modulo (remainder from division):

%

- Exponent:

**

Math with Integers

Operator	Description	Example	Evaluates To
+	Addition	7 + 3	10
-	Subtraction	7 - 3	4
*	Multiplication	7 * 3	21
/	Division (True)	7 / 3	2.333333
//	Division (Integer)	7 // 3	2
%	Modulus	7 % 3	1
**	Exponent	7 ** 3	343

Math with Floats

Operator	Description	Example	Evaluates To
+	Addition	7.0 + 3.0	10.0
-	Subtraction	7.0 - 3.0	4.0
*	Multiplication	7.0 * 3.0	21.0
/	Division (True)	7.0 / 3.0	2.333333
//	Division (Integer)	7.0 // 3.0	2.0
%	Modulus	7.0 % 3.0	1.0
**	Exponent	7.0 ** 3.0	343.0

Order of Operations

- PEMDAS
 - › Step 1: parenthesis
 - › Step 2: exponent
 - › Step 3: multiplication, division, and modulo
 - › Step 4: addition and subtraction

- Prioritize with parenthesis:

```
(cost + tax) * discount  
cost + (tax * discount)
```

- Without parentheses, expression are evaluated according to the order of operations.

Condition

- Syntax

```
if condition:  
    statement1
```

```
if condition:  
    statement1  
else:  
    statement2
```

- Place a colon : after the condition.
- You must indent the lines underneath the **if** statement.
- The **else** is optional.

Example

```
age = int(input("Enter your age: "))
if age >= 18:
    print("You can vote!")
else:
    print("Not yet")
```

Enter your age: 18
You can vote!

Enter your age: 17
Not yet



Indentation
matters!

Comparison Operators

Operator	Meaning	Sample Condition	Evaluates To
<code>==</code>	equal to	<code>5 == 5</code>	<code>True</code>
<code>!=</code>	not equal to	<code>8 != 5</code>	<code>True</code>
<code>></code>	greater than	<code>3 > 10</code>	<code>False</code>
<code><</code>	less than	<code>5 < 8</code>	<code>True</code>
<code>>=</code>	greater than or equal to	<code>5 >= 10</code>	<code>False</code>
<code><=</code>	less than or equal to	<code>5 <= 5</code>	<code>True</code>

Multiple Conditions

- Sometimes we may want different things to happen based on different conditions.
 - › For example, if the score is greater than or equal to 90, the user gets an A, but if it is greater than or equal to 80, the user gets a B, and so on.
- In Python, we can use
if-elif-else

```
if condition:  
    statement1  
elif condition2:  
    statement2  
else:  
    default
```

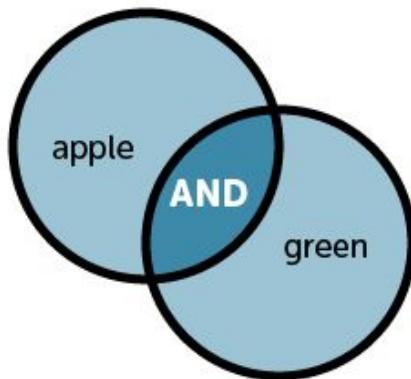
Conditional Statement Rules

- You can have an **if** without **elif** or **else**.
- You can't have **elif** without **if** first.
- You can't have **else** without **if** first.
- You can only have one **if** (in a chain).
- You can only have one **else** (in a chain).
- You can have as many **elif** as you desire (in middle of chain).

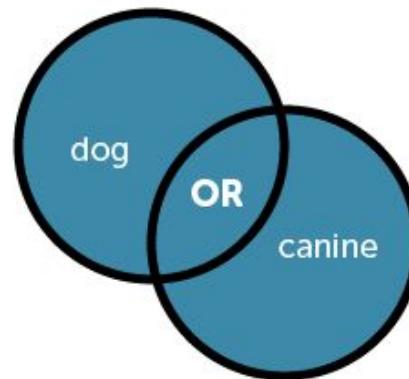


Boolean Logic

- Branching statements let us choose between multiple options.
- Sometimes we want multiple conditions to be true.
- Boolean logic allows us to use **and**, **or**, and **not** operators to make more sophisticated conditions.



You will get results that contain:
both **apple** and **green**



You will get results that contain:
either **dog**, or **canine**, or both



You will get results that contain:
football, and not **soccer**

and Operator

- Both expressions must be **True** for the whole expression to be **True**

expression1 and expression2

```
num = int(input("Enter a number (0-9) : "))
if num >= 0 and num < 10:
    print("You entered a single digit positive number.")
else:
    print("You did not enter a correct number.")
```

```
Enter a number (0-9) : 15
You did not enter a correct number.
```

```
Enter a number (0-9) : 8
You entered a single digit positive number.
```

or Operator

- Only one expression must be **True** for the whole expression to be **True**

expression1 or expression2

```
num = int(input("Enter a number: "))
if num == 8 or num == 24:
    print("You entered Kobe's jersey number.")
```

Enter a number: 15

Enter a number: 8
You entered Kobe's jersey number.

not Operator

- The **not** operator requires one Boolean expression
- The expression must be **False** for the whole expression to be **True**

not *expression*

```
num = int(input("Enter a number: "))

if not num >= 0:
    print("You entered a negative number.")
```

Enter a number: 8

Enter a number: -99
You entered a negative number.

True & False Tables

and

True	True	False
False	False	False

or

True	True	True
False	True	False

not

True	False
False	True