

# Introduction to programming with Python

## Session 2

# Objectives

- Review what we have seen in the previous session:
  - Variables
  - Data types
  - Functions
- Controlling the flow of our programs

# Variables 1: dynamic typing

- Python has strong dynamic typing
  - No need to declare the type of the variable
  - Python recognises the type according to the value of the variable

```
my_variable = 100
print(type(my_variable))    # will print <class 'int'>
my_variable="100"          # notice the quote for a string data type
print(type(my_variable))    # will print <class 'str'>
```

# Variables 2: case sensitive

- Python is case sensitive

```
My_variable = 100
print(id(my_variable))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'my_variable' is not defined
```

# Variables 3: where it is stored

- A variable has an address in memory

# Variables 4: scope

- A variable has a **scope**: only accessible from where it is defined.
- A variable is wiped out from memory once it stops being used. We say that it is **garbage collected**

We define *variable\_a* in *program\_a.py*

```
#program_a.py  
variable_a = 42
```

We try to use *variable\_a* in *program\_b.py*. What is wrong?

```
#program_b.py  
print(variable_a)
```

# Variables 5: naming rules

- A variable name is a non-empty sequence of characters of any length with:
  - The start character can be the underscore "\_" or a capital or lower case letter.
  - Python keywords are not allowed as identifier names!

# Keywords (to not use as variable name)

|       |        |        |        |       |          |        |
|-------|--------|--------|--------|-------|----------|--------|
| and   | as     | assert | break  | class | continue | def    |
| del   | elif   | else   | except | exec  | finally  | for    |
| from  | global | if     | import | in    | is       | lambda |
| not   | or     | pass   | print  | raise | return   | try    |
| while | with   | yield  |        |       |          |        |



# Exercise 1: From algorithm to Python code

- Translate the following algorithm into Python code:
  - Step 1: Use a variable named *miles* with initial value 100.
  - Step 2: Multiply *miles* by 1.609 and assign it to a variable named kilometres
  - Step 3: Display the value of kilometres with the function `print()`

👁 Show solution

## Exercise 2.1: Area of a rectangular room

- Use variables for length, width and area.
- Set length to be 3 and width to be 4
- The multiply operator in Python is the sign\*
- Formulae of the area of a square: length \* width
- Use the **print()** function to display the result

👁 Show solution

## Exercise 2.2: Dynamic Area

- Use the **input()** function to ask values from the user.
  - Ask the user to give the *length* and *width* required
  - Convert the input received into a number with the function **float()**
- 👁 Show solution

# Common Data Types: definition

- Numeric types:
  - **Integer:** whole number

```
type(1)    # <class 'int'>
```

- **Float:** number with decimal

```
type(1.0)  # <class 'float'>
```

- **String**

```
type("1.0") # <class 'str'>
```

# Common Data Types: Examples

| Data type | Examples                                |
|-----------|---|
| Integers  | -2, -1, 0, 1, 2, 3, 4, 5                |
| Floats    | -1.25, -1.0, --0.5, 0.0, 0.5, 1.0, 1.25 |
| Strings   | 'a', 'aa', 'aaa', 'Hello!', '11 cats'   |

# Numeric Operators

| Name | Meaning          | Example      | Result |
|------|------------------|--------------|--------|
| +    | Addition         | $34 + 1$     | 35     |
| -    | Substraction     | $34.0 - 0.1$ | 33.9   |
| *    | Multiplication   | $300 * 30$   | 9000   |
| /    | Float division   | $1 / 2$      | 0.5    |
| //   | Integer Division | $1 // 2$     | 0      |
| **   | Exponentiation   | $4 ** 0.5$   | 2.0    |
| %    | Remainder        | $20 \% 3$    | 2      |

# The % (modulo or remainder) operator (1/2)

$$\begin{array}{r} 2 \\ 3 \overline{) 7} \\ \underline{6} \\ 1 \end{array}$$

$$\begin{array}{r} 3 \\ 4 \overline{) 12} \\ \underline{12} \\ 0 \end{array}$$

$$\begin{array}{r} 3 \\ 8 \overline{) 26} \\ \underline{24} \\ 2 \end{array}$$

Divisor  $\rightarrow$   $\begin{array}{r} 1 \\ 13 \overline{) 20} \\ \underline{13} \\ 7 \end{array}$   $\leftarrow$  Quotient

$\leftarrow$  Dividend

$\leftarrow$  Remainder

# The % (modulo or remainder) operator (2/2)

**Remainder or Modulo** is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd.



# Arithmetic expressions

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

...is translated to:

```
(3 + 4 * x) / 5 - 10 * (y - 5) * (a + b + c) / x + \
9 * (4 / x + (9 + x) / y)
```

NB: the sign \ is an "escaped" character, to break a line for readability

# Exercise: Computing Loan Payments

Let the user enter the yearly interest rate, number of years, and loan amount, and compute monthly payment and total payment.

- Use `input()`
- Translate the following arithmetic expression in Python:

$$\text{monthlyPayment} = \frac{\text{loanAmount} \times \text{monthlyInterestRate}}{1 - \frac{1}{(1 + \text{monthlyInterestRate})^{\text{numberOfYears} \times 12}}}$$

# Solution: Computing Loan Payments

👁 Show solution

# Operations on the String Type (1/2)

## Concatenation

The expression concatenating a string returns a new string:

```
first_string = "abra"
second_string = "cada"
third_string = "bra"
concatenated_string = first_string + second_string \
    + third_string
print("first_string is", first_string,
      "second_string is", second_string,
      "third_string is ", third_string,
      "concatenated_string is ", concatenated_string)
```

# Operations on the String Type (2/2)

## Slicing

Remember that the string is a **sequence** of characters

The items of a sequence can be accessed through indexes

|                       |   |   |   |   |   |   |   |   |   |   |    |
|-----------------------|---|---|---|---|---|---|---|---|---|---|----|
| Items<br>(characters) | a | b | r | a | c | a | d | a | b | r | a  |
| Indexes               | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

Get the first element of the sequence:

```
my_string_variable = "abracadabra"  
first_elem = my_string_variable[0]
```

# Built in functions seen so far

| Input/Ouput | Conversion type: | Introspection: |
|-------------|------------------|----------------|
| input()     | int()            | type()         |
| print()     | float()          | dir()          |
|             | str()            | help()         |
|             |                  | id()           |

All the built in functions:

<https://docs.python.org/3.5/library/functions.html>

# Defining our own function

To define a function, we use the keyword **def**, the name of the function, the brackets, and the colon

Then the body of the function needs to be indented

```
def name_of_the_function():  
    # body of the function
```

When we define a function, python saves it in its memory, but doesn't execute it automatically.

# Calling our own function

To call (or execute or run) a function, we use the name of the function AND the brackets. Without the brackets, the function is not called.

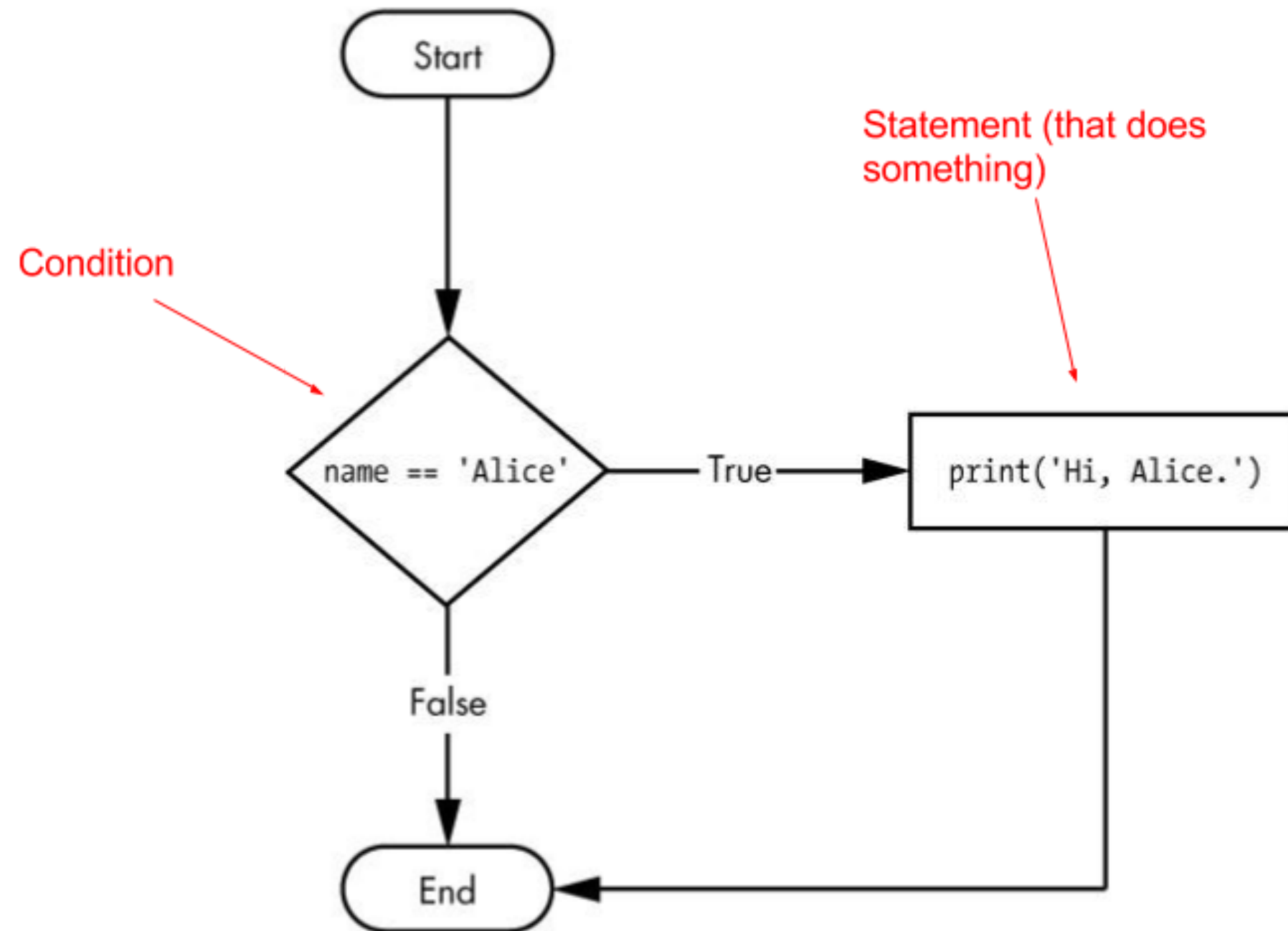
```
name_of_the_function()
```

Notice the difference between defining and calling a function



# Controlling the flow of our programs

We can represent the flow of execution with a flow chart



# Structure of a simple if statement

Pseudo code:

```
if condition:  
    # statement (mind the indentation)
```

Example, representation of the flow chart example in python code:

```
if name=='Alice':  
    print('Hi Alice')
```

# The two-way if statement

Pseudo code:

```
if condition:  
    # statement (mind the indentation)  
else:  
    # statement executed when the condition is False
```

Example, representation of the flow chart example in python code with an else statement:

```
if name=='Alice':  
    print('Hi Alice')  
else:  
    print('Hi')
```

# Multiple Alternative if Statements

## The naive way

```
if condition:
    # statement (mind the indentation)
else:
    if condition:
        # statement executed when
        # the previous condition is False
    else:
        # statement executed when none of
        # the previous condition is verified
```

# Multiple Alternative if Statements

The better way, the pythonic way

```
if condition:
    # statement (mind the indentation)
elif condition:
    # statement executed when
    # the previous condition is False
elif condition:
    # statement executed when none of
    # the previous condition is verified
else:
    # executed when all conditions are False
```

# Value of the condition

The program will execute the statement(s) only if the condition is True.

The condition is actually a **boolean** type.

# The Boolean Type

- Has only 2 possible values: **True** or **False**. Notice that they are both capitalized, which is important because Python is case sensitive
- It is obtained as a result of a comparison expression.

# Comparison Operators

| Operator | Meaning               |
|----------|-----------------------|
| <        | less than             |
| <=       | less than or equal    |
| >        | greater than          |
| >=       | greater than or equal |
| ==       | equal to              |
| !=       | not equal to          |



# Examples

trinket

Python3

>\_ Run

Share

Remix

< >

main.py

+ ↕

```
1 'hello' == 'hello'
2 'hello' == 'Hello'
3 'dog' != 'cat'
4 True == True
5 True != False
6 42 == 42.0
7 42 == '42'
```

Powered by trinket

```
>>> 'hello' == 'hello'
>>> 'hello' == 'Hello'
True
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
>>> 42 == '42'
False
>>> 
```

# Difference between '==' and '='

- The sign = is the sign of **assignment**, it is used for assigning a value to a variable
- The sign == is the sign of **comparison**, it compares 2 values and returns a boolean (True or False)

# Exercise: password

Create a program that asks the user for a password.

- Have the password defined in your program, in a variable called "PASSWORD"
- Use input() to receive the password entered by the user
- If the word entered by the user matches the password, display "Access Granted", otherwise "Forbidden"

# Solution: password

👁 Show solution

# Truth tables

Show every possible result of a Boolean operator.

The **and** Operator's Truth Table

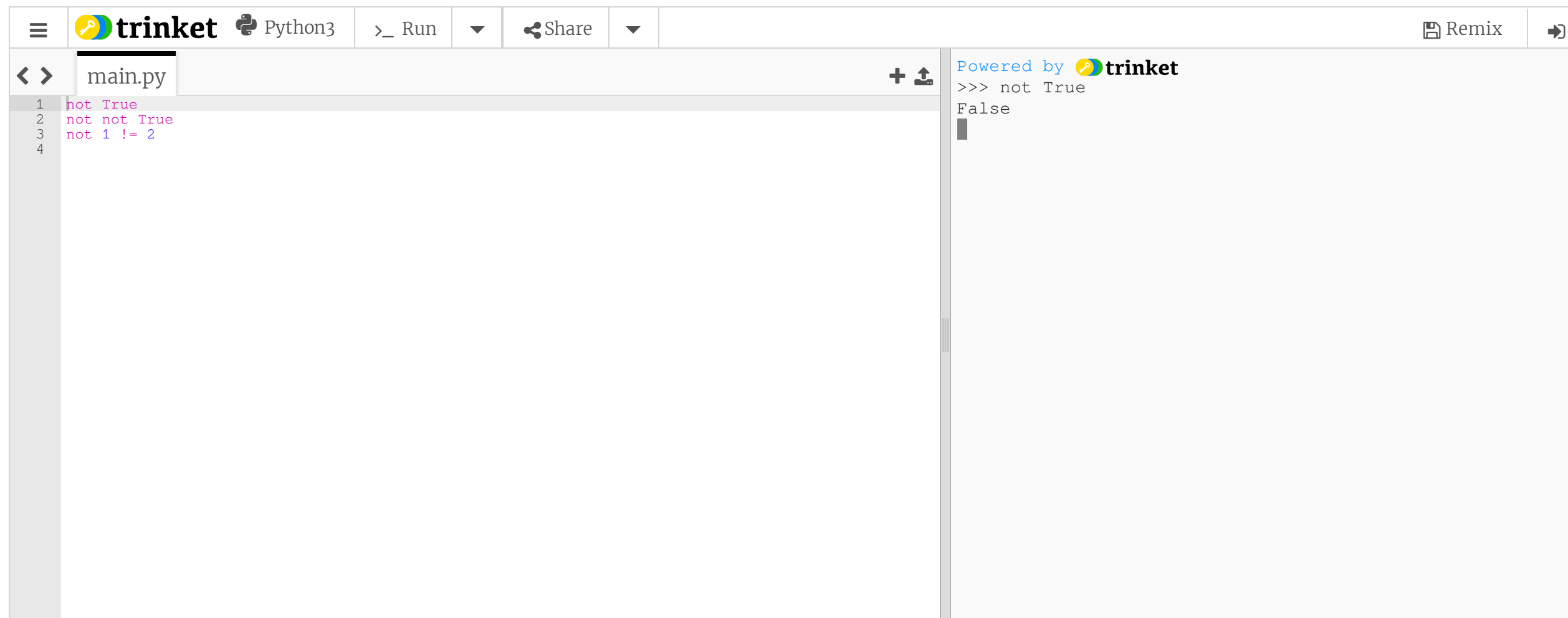
| Expression      | Evaluates to... |
|-----------------|-----------------|
| True and True   | True            |
| True and False  | False           |
| False and True  | False           |
| False and False | False           |

## The **or** Operator's Truth Table

| Expression     | Evaluates to... |
|----------------|-----------------|
| True or True   | True            |
| True or False  | True            |
| False or True  | True            |
| False or False | False           |

# The **not** Operator

It operates on only one Boolean value (or expression). The not operator simply evaluates to the opposite Boolean value.



The screenshot shows the Trinket Python3 IDE interface. The top bar includes the Trinket logo, 'Python3', a 'Run' button, a 'Share' button, and a 'Remix' button. The left pane shows a file named 'main.py' with the following code:

```
1 not True
2 not not True
3 not 1 != 2
4
```

The right pane shows the output of the code, indicating it is 'Powered by trinket' and displaying the results of the expressions:

```
>>> not True
False
```

# Exercise: password and login

Create a program that ask the user for a login and password.

- Have the password "PASSWORD" AND login "LOGIN" defined in your program, in corresponding variables
- Use input() to receive the password and login entered by the user
- If login and password match the values of your PASSWORD and LOGIN, display "Access Granted", else, "Forbidden"



# Solution: password and login

👁 Show solution

# Exercise: check number divisor

Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

- Use `input()` take the user input
- Use `int()` to convert the value return by input into an integer
- Use `%` to see if a number `x` is divisible by an other number `y`, if `x%y` returns 0, then `x` is divisible by `y`
- Use `print()`

# Solution: control flow

👁 Show solution

# Exercise: grading students

Write a program that is going to give the grade of a student according to the score obtained.

- Display 'A' if the score is greater than 90
- Display 'B' if the score is between 80 and 90
- Display 'C' if the score is between 70 and 80
- Display 'D' if the score is between 60 and 70
- Display 'F' if the score is lower than 60

# Solution: grading students

👁 Show solution

# Exercise: determining a leap year

This program first prompts the user to enter a year as an int value and checks if it is a leap year.

A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

- Use `input()` to take the user input (the year, i.e. 2016) and convert it with `int()`
- Use `%` to see if a number `x` is divisible by another number `y`, if `x%y` returns 0, then `x` is divisible by `y`
- Check if the year is divisible by 4 AND not divisible by 100
- OR check if the year is divisible by 400.
- Use `print()`

# Solution: determining a leap year

👁 Complete solution

# Solution optimized: determining a leap year

- 👁 Condition to use
- 👁 Complete solution



# Exercise: Chinese Zodiac sign

Now let us write a program to find out the Chinese Zodiac sign for a given year. The Chinese Zodiac sign is based on a **12-year cycle**, each year being represented by an animal: rat, ox, tiger, rabbit, dragon, snake, horse, sheep, monkey, rooster, dog, and pig, in this cycle

👁 Hint 1

👁 Hint 2

# Exercise: Chinese Zodiac sign

| Year | Zodiac sign |
|------|-------------|
| 0    | monkey      |
| 1    | rooster     |
| 2    | dog         |
| 3    | pig         |
| 4    | rat         |
| 5    | ox          |
| 6    | tiger       |
| 7    | rabbit      |
| 8    | dragon      |
| 9    | snake       |
| 10   | horse       |
| 11   | sheep       |

👁 Complete solution