Co-funded by the
Erasmus+ Programme
of the European Union

PYTHON

ENGLISH

Introduction

This Python tutorial uses IDLE, the integrated development environment that comes bundled with the default implementation of the language which can be downloaded from www.python.org.
In Python the hash character (#) is used to insert comments, when Python encounters a hash character it will ignore the rest of the line.

Variables

In Python a variable does not need to be declared, it is created the first time a value is assigned to it. It also does not need to be created as a particular type. Values are assigned to variables by using the equal sign (=).

```
>>> myFirstVariable = 5        # Integer
>>> mySecondVariable = 3.14    # Float
>>> myThirdVariable = "abc"    # String
```

Multiple variables can be created in one line.

```
>>> x, y = 5,3                 # x = 5 and y = 3
```

Arithmetic operators

'Basic mathematical operators are built into the Python programming language.

```
>>> # Create variables
>>> x, y = 5,3
>>> print("x =", x, "and y =",y)
x = 5 and y = 3
>>> # Addition
>>> z = x + y
>>> print("z =",z)
z = 8
>>> # Subtraction
>>> z = x-y
>>> print("z =",z)
z = 2
```

```
>>> # Multiplication
>>> z = x*y
>>> print("z =",z)
z = 15
>>> # Division
>>> z = x/y
>>> print("z =",z)
z = 1.6666666666666667
>>> # Exponentiation (Power)
>>> z = x**y
>>> print("z =",z)
z = 125
```

Mathematical functions are generally not built into Python so to use them we must first import the "math" module. We then call the functions with the prefix "math."

```
>>> import math
>>> a = 169
>>> b = math.sqrt(a)
>>> print("The square root of",a,"is",b)
The square root of 169 is 13.0
```

It should be noted that in Python, like other programming languages, floating point arithmetic may seem inaccurate. This is due to the way floating point variables are represented. For example, if I assign the value 0.1 to a variable it will in fact not be stored as exactly 1/10. This can produce results that seem inaccurate but the errors are so small that it's mostly just annoying and we will ignore it in this tutorial.

```
>>> x=3
>>> y=0.1
>>> print(x*y)
0.30000000000000004
```

Comparison operators

To compare values we use comparison operators.

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

Conditional statements

To make decisions based on conditions we can use the „if“ statements.

```
>>> x,y = 5,3
>>> if x > y:
      print(x, "is greater than", y)


5 is greater than 3
```

Note that Python uses indentation to group statements. After typing „if x > y:“, the following command lines that are to be executed if the condition is true are indented. To exit the “if” statement press return twice after the last command. Not having the command lines inside the “if” statement indented will cause an error.

```
>>> if x > y:
print(x, "is greater than", y)
SyntaxError: expected an indented block
```

If we want to do something else if the condition is not met we use the “else” keyword.

```
>>> x, y = 5,3
>>> if y > x:
      print(y, "is greater than", x)
else:
      print(y, "is not greater than", x)


3 is not greater than 5
```

Note again the indentation, the “else” keyword must have the same indentation as the “if” keyword it is associated with. After entering the last command to be executed if the statement is met, press backspace. We can also have additional conditions using the “elif” keyword.

```
>>> x, y = 4,4
>>> if x > y:
      print(x, "is greater than", y)
elif y > x:
      print(y, "is greater than", x)
else:
      print(x, "is equal to", y)


4 is equal to 4
```

Loops and sequences

To repeat the same set of commands we can use the "for" keyword. The "for" loop iterates over a sequence and uses an iterator (variable) to store the current value from the sequence. The syntax of the "for" loop is: for [iterator] in [sequence]:

It is useful to know about the "range()" function when creating "for" loops. The "range()" function creates a sequence of numbers, range(*m*) creates a sequence of *m* numbers from *0* to *m-1*.

```
>>> for i in range(3):
      print(i)


0
1
2
```

Like in the "if" statements indentation is used to group statements and to close the "for" loop we press return twice after the last command.

The "range()" function can be given a different starting value, range(*n*, *m*) creates a sequence of numbers from *n* to *m-1*. It can also be given a different increment value, range(*n*, *m*, *q*) creates a sequence of numbers from *n* to *m-1* incrementing by *q* between numbers. All values passed to the "range()" function must be integers.

The sequence to be iterated over can also be created before the "for" loop.

```
>>> myFirstSequence = range(2,5)
>>> for i in myFirstSequence:
      print(i, "is in sequence")


2 is in sequence
3 is in sequence
4 is in sequence
```

```
>>> mySecondSequence = range(4,10,2)
>>> for i in mySecondSequence:
      print(i, "is in sequence")


4 is in sequence
6 is in sequence
8 is in sequence
```

Other types of sequences the "for" loop can iterate over include strings and lists.

```
>>> myString = "GBPE"
>>> for i in myString:
      print(i,"is in 'GBPE' ")


G is in 'GBPE'
B is in 'GBPE'
P is in 'GBPE'
E is in 'GBPE'
```

```
>>> myList = ["CZ","IS","IT","PT","ES"]
>>> for i in myList:
      print(i,"is in GBPE")
```

```
CZ is in GBPE
IS is in GBPE
IT is in GBPE
PT is in GBPE
```

```
ES is in GBPE
```

Functions

To define functions in Python we use the “def” keyword. Like before Python uses indentation to group commands and to complete the definition of the function press return twice after the last command.

```
>>> def sayHello():
      print("Hello world!")


>>> sayHello()
Hello world!
```

The parenthesis after the name of the function is necessary both when defining the function and when calling it. If we want to pass arguments to the function we would indicate it in the parenthesis. It is possible to define the same function for different numbers of arguments, when the function is called Python will use the definition that matches the number of arguments being passed.

```
>>> myName = "Grétar"
>>> def sayHello(name):
      print("Hello",name)


>>> sayHello(myName)
Hello Grétar
```

```
>>> myCountry = "Iceland"
>>> def sayHello (name, country):
      print("Hello",name,"from",country)


>>> sayHello(myName, myCountry)
Hello Grétar from Iceland
```

Special care must be taken with variables when it comes to functions. In this tutorial the following rules will apply to variables:

Variables created inside a function can only be used inside the function.

Variables created outside a function can be accessed inside the function but not assigned a new value. Trying to assign a new value inside a function to a variable created outside the function will either create a new variable with the same name inside the function or cause an error.

```
>>> def insideVar():
      inVar = "inside"
      print(inVar)


>>> insideVar()
inside
>>> print(inVar)
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    print(inVar)
NameError: name 'inVar' is not defined
```

```
>>> outVar = "outside"
>>> def changeOutsideVar():
      outVar = "inside"
      print(outVar)


>>> changeOutsideVar()
inside
>>> print(outVar)
outside
```

There are ways to get around these rules but that is beyond the scope of this tutorial

User input

To ask for input from the user we can use the “input()” function. When the “input()” function is called Python will wait until the user responds.

```
>>> def askForName():
      name = input("Enter your name: ")
      print("Hello",name)


>>> askForName()
Enter your name: Grétar
Hello Grétar
```

By default the “input()” function returns a string so if we want the user to submit a number we must convert the type of the response.

```
>>> def askForNumber():
      returnedString = input("Enter number to be squared: ")
      returnedNumber = float(returnedString)
      print("Your number squared is",returnedNumber**2)


>>> askForNumber()
Enter number to be squared: 7
Your number squared is 49.0
```

In this case I converted the response to a floating point number but I could also have converted it to an integer by using “int()” instead of “float()”. Note that if the response is incompatible with the type you try to convert into it will cause an error.

The user input can be converted immediately (without first being stored)

```
>>> def askForAnotherNumber():
      returnedNumber = float(input("Enter number to be squared: "))
      print("Your number squared is",returnedNumber**2)


>>> askForAnotherNumber()
Enter number to be squared: 8
Your number squared is 64.0
```

Example 1

Enter the radius of the circle and calculate the content and circuit of the circle.

Enter a radius of 5 m.
Result: The content is 78.5. The circuit is 31.4.

```
>>> def circle():
      radius = int(input("Enter radius: "))
      print("The area is:",radius**2*3.14)
      print("The circumference is:",2*radius*3.14)


>>> circle()
Enter radius: 5
The area is: 78.5
The circumference is: 31.400000000000002
```

Example 2

Enter the length and width of the land and the price for 1m^2. Calculate and display the area of land and the price of the land.
Enter length 50 m and width 70 m and price 0,5 euro / m^2. Content is 3500 m2. The price is 7000 euros.

```
>>> def land():
      length = float(input("Enter length (m): "))
      width = float(input("Enter width (m): "))
      unitprice = float(input("Enter unit price (€/m^2): "))
      area = length*width
      print("The area is: ",area,"squaremeters")
      print("The price is: ",area*unitprice,"€")
>>> land()
Enter length (m): 50
Enter width (m): 70
Enter unit price (€/m^2): 0.5
The area is:  3500.0 squaremeters
The price is:  1750.0 €
```

Examples 3

Enter price and quantity. Calculate and Display VAT (20%), Price with VAT and Sales (Price with VAT * pcs).
For example price 100, pcs 20.
VAT is 20, Price with VAT 120 and Sales 2400.

```
>>> def vatCalc():
      vat = 0.2
      price = float(input("Enter price (€): "))
      quantity = float(input("Enter quantity: "))
      print("The VAT is",vat*price, "€")
      print("The price with VAT is",vat*price+price,"€")
      print("The sales total is",(vat*price+price)*quantity,"€")
```

```
>>> vatCalc()
Enter price (€): 100
Enter quantity: 20
The VAT is 20.0 €
The price with VAT is 120.0 €
The sales total is 2400.0 €
```

Examples 4

Enter time and track. Calculates speed in meter/second (m/s) and kilometer/hour (km/h).

```
>>> def speed():
      time = float(input("Enter time (s): "))
      distance = float(input("Enter distance (m): "))
      speedms = distance/time
      speedkh = speedms*3.6
      print("The speed is",speedms,"m/s or",speedkh,"km/h")
```

IF

Examples 5

Enter the number and decide if is it positive or negative. (zero is positive)

```
>>> def posOrNeg():
      number = float(input("Enter number: "))
      if number < 0:
            print(number,"is a negative number")
      else:
            print(number,"is a positive number")
```

Examples 6

Enter side A and side B. Deside if it is a square A = B or a rectangle and calculate the content of the shape.

Try A = 6 and B = 7. This is a rectangle. Content is 42.

Try A = 6 and B = 6. This is a square. Content is 36.

```
>>> def squareOrRect():
      A = float(input("Enter side A: "))
      B = float(input("Enter side B: "))
      area = A*B
      if A == B:
            print("This is a square with area",area)
      else:
            print("This is a rectangle with area",area)


>>> squareOrRect()
Enter side A: 6
Enter side B: 7
This is a rectangle with area 42.0
```

```
>>> squareOrRect()
Enter side A: 6
Enter side B: 6
This is a square with area 36.0
```

Examples 7

Enter the number X and decide if it is between 5 and 50.

Help:
If A <= 5 and Number <= 50 Then
Try entering number 2. Number is not in interval
Try to enter the number 40. The number is in the interval.

```
>>> def interval():
      X = float(input("Enter number: "))
      if X > 5 and X < 50:
            print("The number is in the interval")
      else:
            print("The number is not in the interval")


>>> interval()
Enter number: 2
The number is not in the interval
>>> interval()
Enter number: 40
The number is in the interval
```

Examples 8

Enter two numbers A and B. If A is greater than B, calculate the difference and store it in variable C. If the numbers are equal, multiply the numbers and store in variable C. If A is less than B, calculate their sum. Display the variable C.

```
>>> def multi():
      A = float(input("Enter number A: "))
      B = float(input("Enter number B: "))
      if A > B:
            C = A - B
      elif A == B:
            C = A*B
      elif A < B:
            C = A + B
      print("C =",C)
```

Examples 9

Enter the number of produced pieces (P) and the price per piece (S) and calculate the amount of payroll tax according to the table. Displays salary and tax.

Salary employee

0- 2500 Euro 20%

from 2500 - 5000 Euro 15%

up to 5000 Euro 10%

```
>>> def payrolltax():
      P = float(input("Enter number of produced pieces: "))
      S = float(input("Enter the price per piece (€): "))
      salary = P*S
      if salary <= 2500:
            tax = salary*0.2
      elif salary <= 5000:
            tax = 2500*0.2+(salary - 2500)*0.15
      elif salary > 5000:
            tax = 2500*0.2+(5000 - 2500)*0.15+(salary - 5000)*0.1
      print("Total salary is",salary,"€")
      print("Payroll tax is",tax,"€")
      print("Net salary is",salary - tax,"€")
```

Examples 10

Enter the speed and find out what penalty the driver will pay.

50-70 km / h ..... 50 euro
71 - 90 km / h... 100 euro
Over 90 km / h... 200 euro

```
>>> def speedingticket():
      speed = float(input("Enter the speed (km/h): "))
      if speed < 50:
            penalty = 0
      elif speed < 71:
            penalty = 50
      elif speed < 91:
            penalty = 100
      else:
            penalty = 200
      print("The penalty for driving at",speed,"km/h is",penalty,"€")
```

Cycle

Examples 11

Enter 5 numbers. Sum of the numbers to display.

Hint: When using a loop to create a result (like a sum) it is necessary to initialize the variable before starting the loop.

```
>>> def sumNum():
      theSum = 0
      for i in range(5):
            userInput = float(input("Enter number: "))
            theSum += userInput
      print("The sum is",theSum)
```

Examples 12

Enter 4 times. Calculate and view average time.

```
>>> def avTime():
      theSum = 0
      for i in range(4):
            userInput = float(input("Enter time (s): "))
            theSum += userInput
      averageTime = theSum/4
      print("The average time is",averageTime,"s")
```

Examples 13

For numbers 1 to 5, display the second power of the number.

```
>>> def squared():
      for i in range(1,6):
            print(i,"squared is",i**2)
```

Examples 14

Find out how many of the 8 integers entered are greater than 0

```
>>> def gr8():
      greater = 0
      for i in range(8):
            userInput = int(input("Enter integer: "))
            if userInput > 0:
                  greater += 1
      print("There are",greater,"integers greater than 8")
```

Examples 15

Enter 6 attempts by the athlete to throw the disc and find out which was the longest.

```
>>> def discus():
      bestattempt = 0
      for i in range(6):
            userInput = float(input("Enter attempt (m): "))
            if userInput > bestattempt:
                  bestattempt = userInput
      print("The best attempt was ",bestattempt,"m")
```

Examples 16

Create a program that generates 5 numbers (numbers from 1 to 49). Display the numbers.
Hint: To generate pseudo-random numbers you can use the “random” module.

```
>>> def ranum():
      import random
      for i in range(5):
            number= random.randrange(1,49)
            print(number)
```

Examples 17

Generate 10 numbers from 1 to 6 to view and see how many times the number "6" was generated.

```
>>> def ran6():
      import random
      sixes = 0
      for i in range(10):
            number = random.randrange(1,7)
            print(number)
            if number == 6:
                  sixes += 1
      print("There are",sixes,"sixes")
```

CZECH

Úvod

Tento kurz Pythonu používá IDLE, integrované vývojové prostředí, které je dodáváno s výchozí implementací jazyka, kterou lze stáhnout z www.python.org.V Pythonu se znak hash (#) používá k vkládání komentářů, když Python narazí na znak hash, bude ignorovat zbytek řádku.

Proměnné

V Pythonu nemusí být proměnná deklarována, je vytvořena při prvním přiřazení hodnoty. Také nemusí být vytvořen jako konkrétní typ. Hodnoty jsou přiřazeny proměnným pomocí znaménka rovná se (=).

```
>>> myFirstVariable = 5        # Integer
>>> mySecondVariable = 3.14    # Float
>>> myThirdVariable = "abc"    # String
```

V jednom řádku lze vytvořit více proměnných.

```
>>> x, y = 5,3                 # x = 5 and y = 3
```

Aritmetické operátory

Základní matematické operátory jsou integrovány do programovacího jazyka Python.

```
>>> # Create variables
>>> x, y = 5,3
>>> print("x =", x, "and y =",y)
x = 5 and y = 3
>>> # Addition
>>> z = x + y
>>> print("z =",z)
z = 8
>>> # Subtraction
>>> z = x-y
>>> print("z =",z)
z = 2
```

```
>>> # Multiplication
>>> z = x*y
>>> print("z =",z)
z = 15
>>> # Division
>>> z = x/y
>>> print("z =",z)
z = 1.6666666666666667
>>> # Exponentiation (Power)
>>> z = x**y
>>> print("z =",z)
z = 125
```

Matematické funkce obecně nejsou zabudovány do Pythonu, takže pro jejich použití musíme nejprve importovat modul "math". Funkce pak nazýváme předponou "math".

```
>>> import math
>>> a = 169
>>> b = math.sqrt(a)
>>> print("The square root of",a,"is",b)
The square root of 169 is 13.0
```

Je třeba poznamenat, že v Pythonu, stejně jako v jiných programovacích jazycích, se aritmetika s plovoucí desetinnou čárkou může zdát nepřesná. To je způsobeno způsobem, jakým jsou reprezentovány proměnné s plovoucí desetinnou čárkou. Pokud například přiřadím proměnné hodnotu 0,1, nebude ve skutečnosti uložena přesně jako 1/10. To může vést k výsledkům, které se zdají být nepřesné, ale chyby jsou tak malé, že je to většinou jen nepříjemné a v tomto tutoriálu je budeme ignorovat.

```
>>> x=3
>>> y=0.1
>>> print(x*y)
0.30000000000000004
```

Porovnávací operátory

Pro porovnání hodnot používáme porovnávací operátory.

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

Podmíněné výroky

Abychom se mohli rozhodovat na základě podmínek, můžeme použít příkazy "IF".

```
>>> x,y = 5,3
>>> if x > y:
      print(x, "is greater than", y)


5 is greater than 3
```

Všimněte si, že Python používá odsazení k seskupení příkazů. Po zadání "if x > y:" jsou odsazeny následující příkazové řádky, které mají být provedeny, pokud je podmínka pravdivá. Chcete-li ukončit příkaz "if", stiskněte klávesu return dvakrát po posledním příkazu. Pokud nebudou příkazové řádky uvnitř příkazu „if“ odsazeny, dojde k chybě.

```
>>> if x > y:
print(x, "is greater than", y)
SyntaxError: expected an indented block
```

Pokud chceme při nesplnění podmínky udělat něco jiného, použijeme klíčové slovo „else“.

```
>>> x, y = 5,3
>>> if y > x:
      print(y, "is greater than", x)
else:
      print(y, "is not greater than", x)


3 is not greater than 5
```

Všimněte si znovu odsazení, klíčové slovo „else“ musí mít stejné odsazení jako klíčové slovo „if“, se kterým je spojeno. Po zadání posledního příkazu, který se má provést, pokud je příkaz splněn, stiskněte backspace.
Můžeme mít také další podmínky pomocí klíčového slova „elif“.

```
>>> x, y = 4,4
>>> if x > y:
      print(x, "is greater than", y)
elif y > x:
      print(y, "is greater than", x)
```

```
else:
      print(x, "is equal to", y)


4 is equal to 4
```

Smyčky a sekvence

Pro opakování stejné sady příkazů můžeme použít klíčové slovo „for“. Smyčka „for“ iteruje (opakuje určitý proces) sekvenci a používá iterátor (proměnnou) k uložení aktuální hodnoty ze sekvence.
Syntaxe cyklu „for“ je: for [iterátor] v [sekvence]:
Při vytváření smyček „for“ je užitečné vědět o funkci „range()“. Funkce „range()“ vytvoří sekvenci čísel, range(m) vytvoří sekvenci m čísel od 0 do m-1.

```
>>> for i in range(3):
      print(i)


0
1
2
```

Stejně jako v příkazech „if“ se používá odsazení pro seskupení příkazů a pro uzavření cyklu „for“ stiskneme dvakrát return po posledním příkazu.
Funkce „range()“ může mít jinou počáteční hodnotu, range(n, m) vytváří posloupnost čísel od n do m-1. Lze mu také zadat jinou hodnotu přírůstku, range(n, m, q) vytvoří sekvenci čísel od n do m-1 zvyšující se o q mezi čísly. Všechny hodnoty předané funkci „range()“ musí být celá čísla.
Sekvenci, která má být iterována, lze také vytvořit před smyčkou „for“.

```
>>> myFirstSequence = range(2,5)
>>> for i in myFirstSequence:
      print(i, "is in sequence")


2 is in sequence
3 is in sequence
4 is in sequence
```

```
>>> mySecondSequence = range(4,10,2)
>>> for i in mySecondSequence:
      print(i, "is in sequence")


4 is in sequence
6 is in sequence
8 is in sequence
```

Jiné typy sekvencí, přes které může smyčka „for“ iterovat, zahrnují řetězce a seznamy.

```
>>> myString = "GBPE"
>>> for i in myString:
      print(i,"is in 'GBPE' ")


G is in 'GBPE'
B is in 'GBPE'
P is in 'GBPE'
E is in 'GBPE'
```

```
>>> myList = ["CZ","IS","IT","PT","ES"]
>>> for i in myList:
      print(i,"is in GBPE")


CZ is in GBPE
IS is in GBPE
IT is in GBPE
PT is in GBPE
ES is in GBPE
```

Funkce

K definování funkcí v Pythonu používáme klíčové slovo „def“. Stejně jako předtím Python používá odsazení k seskupování příkazů, tak dokončení definice funkce stiskněte dvakrát return po posledním příkazu.

```
>>> def sayHello():
      print("Hello world!")

>>> sayHello()
Hello world!
```

Závorka za názvem funkce je nutná jak při definici funkce, tak při jejím volání. Pokud chceme funkci předat argumenty, uvedli bychom to v závorce.

```
>>> myName = "Grétar"
>>> def sayHello(name):
      print("Hello",name)


>>> sayHello(myName)
Hello Grétar
```

```
>>> myCountry = "Iceland"
>>> def sayHello (name, country):
      print("Hello",name,"from",country)


>>> sayHello(myName, myCountry)
Hello Grétar from Iceland
```

Pokud jde o funkce, je třeba věnovat zvláštní pozornost proměnným. V tomto tutoriálu budou pro proměnné platit následující pravidla:
Proměnné vytvořené uvnitř funkce lze použít pouze uvnitř funkce.
K proměnným vytvořeným mimo funkci lze přistupovat uvnitř funkce, ale nelze jim přiřadit novou hodnotu. Pokus o přiřazení nové hodnoty uvnitř funkce proměnné vytvořené mimo funkci buď vytvoří novou proměnnou se stejným názvem uvnitř funkce, nebo způsobí chybu.

```
>>> def insideVar():
      inVar = "inside"
      print(inVar)


>>> insideVar()
inside
>>> print(inVar)
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in
<module>
    print(inVar)
NameError: name 'inVar' is not
defined
```

```
>>> outVar = "outside"
>>> def changeOutsideVar():
      outVar = "inside"
      print(outVar)


>>> changeOutsideVar()
inside
>>> print(outVar)
outside
```

Existují způsoby, jak tato pravidla obejít, ale to je nad rámec tohoto návodu.

Uživatelský vstup

Chcete-li požádat o vstup od uživatele, můžeme použít funkci „input()“.

```
>>> def askForName():
      name = input("Enter your name: ")
      print("Hello",name)

>>> askForName()
Enter your name: Grétar
Hello Grétar
```

Funkce „input()“ standardně vrací řetězec, takže pokud chceme, aby uživatel odeslal číslo, musíme převést typ odpovědi.

```
>>> def askForNumber():
      returnedString = input("Enter number to be squared: ")
      returnedNumber = float(returnedString)
      print("Your number squared is",returnedNumber**2)


>>> askForNumber()
Enter number to be squared: 7
Your number squared is 49.0
```

V tomto případě jsem převedl odpověď na číslo s pohyblivou řádovou čárkou, ale mohl jsem ji také převést na celé číslo pomocí „int()“ místo „float()“. Všimněte si, že pokud je odpověď nekompatibilní s typem, který se na ni pokoušíte převést, způsobí chybu.
Vstup uživatele lze převést okamžitě (bez předchozího uložení).

```
>>> def askForAnotherNumber():
      returnedNumber = float(input("Enter number to be squared: "))
      print("Your number squared is",returnedNumber**2)


>>> askForAnotherNumber()
Enter number to be squared: 8
Your number squared is 64.0
```

PORTUGAL

Introdução

Este tutorial Python usa o IDLE, o ambiente de desenvolvimento integrado que vem agregado com a implementação padrão da linguagem que pode ser descarregada a partir de www.python.org.

Em Python o caráter cardinal (#) é usado para inserir comentários, quando o Python encontra o caráter cardinal (#), ignorará o resto da linha.

Variáveis

Em Python não é necessário declarar as variáveis, estas são criadas na primeira vez que em que lhes é tribuído um valor. Também não é necessário criar como um tipo específico. Os valores são atribuídos as variáveis utilizando o sinal igual (=).

```
>>> myFirstVariable = 5          # Integer
>>> mySecondVariable = 3.14      # Float
>>> myThirdVariable = "abc"      # String
```

Podem ser criadas várias **variáveis** numa única linha.

```
>>> x, y = 5,3                   # x = 5 and y = 3
```

Operadores aritméticos

Os operadores matemáticos básicos são integrados na linguagem de programação Python.

```
>>> # Create variables
>>> x, y = 5,3
>>> print("x =", x, "and y =",y)
x = 5 and y = 3
>>> # Addition
>>> z = x + y
>>> print("z =",z)
z = 8
>>> # Subtraction
>>> z = x-y
>>> print("z =",z)
z = 2
```

```
>>> Multiplicação
>>> z = x*y
impressão >>>("z =",z)
z = 15
 >>> # Divisão
>>> z = x/y
impressão >>>("z =",z)
z = 1.6666666666666666666666667
>>> # Exponenciação (Poder)
>>> z = x**y
impressão >>>("z =",z)
z = 125
```

As funções matemáticas geralmente não são incorporadas em Python, por isso, para as usar, temos primeiro de importar o módulo "math". Em seguida, chamamos as funções com o prefixo "math".

```
>>> import math
>>> a = 169
>>> b = math.sqrt(a)
>>> print("The square root of",a,"is",b)
The square root of 169 is 13.0
```

Note-se que em Python, como outras linguagens de programação, a aritmética de virgula flutuante pode parecer imprecisa. Isto deve-se à forma como as variáveis de virgula flutuante são representadas. Por exemplo, se eu atribuir o valor 0.1 a uma variável, na verdade não será armazenado como exatamente 1/10. Isto pode produzir resultados que parecem imprecisos, mas os erros são tão pequenos que é principalmente irritante e vamos ignorá-lo neste tutorial.

```
>>> x=3
>>> y=0.1
>>> print(x*y)
0.30000000000000004
```

Operadores de comparação

Para comparar valores usamos operadores de comparação.

Operador	Significado	Exemplo
==	Igual	x == y
!=	Não é igual	x != y
>	Maior do que	x > e
>=	Maior ou igual	x >= y
<	Menos de	x < y
<=	Menos ou igual	x <= y

Declarações condicionais

Para tomar decisões com base em condições podemos usar as declarações "**if**".

```
>>> x,y = 5,3
>>> if x > y:
        print(x, "is greater than", y)


5 is greater than 3
```

Note-se que Python usa **indentação** para declarações de grupo. Depois de escrever "se x > y:", as seguintes linhas de comando que devem ser executadas se a condição for verdadeira são indentadas. Para sair da declaração "se" a imprensa retorna duas vezes após o último comando. *Não ter as linhas de comando dentro da declaração "se" indentada causará um erro.*

```
>>> if x > y:
print(x, "is greater than", y)
SyntaxError: expected an indented block
```

Se quisermos fazer outra coisa se a condição não for cumprida, usamos a palavra-chave "**else**".

```
>>> x, y = 5,3
>>> if y > x:
        print(y, "is greater than", x)
else:
        print(y, "is not greater than", x)


3 is not greater than 5
```

Note novamente a indentação da palavra-chave "**else**" deve ter a mesma indentação que a palavra-chave "**if**" a que está associada. Depois de entrar no último comando a ser executado se a declaração for atendida, pressione o backspace.
Também podemos ter condições adicionais usando a palavra-chave "**elif**".

```
>>> x, y = 4,4
>>> if x > y:
        print(x, "is greater than", y)
elif y > x:
        print(y, "is greater than", x)
else:
        print(x, "is equal to", y)

4 is equal to 4
```

Loops e sequências
Para repetir o mesmo conjunto de comandos, podemos usar a palavra-chave "**for**". O loop "**for**" itera sobre uma sequência e utiliza um iterador (variável) para armazenar o valor atual a partir da sequência. A sintaxe do loop "**for**" é: **for** [iterador] **in** [sequência]:

É útil saber sobre a função "**range()**" ao criar loops "**for**". A função "**range()**" cria uma sequência de números, o **range(*m*)** cria uma sequência de números m de *0* a *m-1*.

```
>>> for i in range(3):
        print(i)


0
1
2
```

Tal como nas declarações "**if**" a indentação é usada para agrupar declarações e fechar, no ciclo "**for**" devemos pressionar *return* duas vezes após o último comando.
A função "**range()**" pode dada a um valor inicial diferente**, range(*n*, *m*)** cria uma sequência de números de ***n*** a ***m-1***. Também pode ser atribuido um valor incremento diferente, **range (*n*, *m*, *q*)** cria

uma sequência de números de n a ***m-1*** **incrementando** por ***q*** entre números. Todos os valores passados para a função "**range()**" devem ser **inteiros**.
A sequência a ser iterada também pode ser criada antes do loop "**for**".

```
>>> myFirstSequence = range(2,5)
>>> for i in myFirstSequence:
        print(i, "is in sequence")


2 is in sequence
3 is in sequence
4 is in sequence
```

```
>>> mySecondSequence = range(4,10,2)
>>> for i in mySecondSequence:
        print(i, "is in sequence")


4 is in sequence
6 is in sequence
8 is in sequence
```

Outros tipos de sequências que o loop "**for**" pode iterar incluem **strings** e **listas**.

```
>>> myString = "GBPE"
>>> for i in myString:
        print(i,"is in 'GBPE' ")


G is in 'GBPE'
B is in 'GBPE'
P is in 'GBPE'
E is in 'GBPE'
```

```
>>> myList = ["CZ","IS","IT","PT","ES"]
>>> for i in myList:
        print(i,"is in GBPE")

CZ is in GBPE
IS is in GBPE
IT is in GBPE
PT is in GBPE
ES is in GBPE
```

Funções
Para definir funções em Python usamos a palavra-chave "**def**". Como anteriormente em Python usamos a indentação para os comandos de grupo e para completar a definição da função, e pressionar return duas vezes após o último comando.

```
>>> def sayHello():
        print("Hello world!")


>>> sayHello()
Hello world!
```

O parêntese após o nome da função é necessário tanto quando se declara a função como quando se chama a função. Se quisermos passar argumentos para a função, devemos indicalos dentros dos parantesês. É possível definir a mesma função para diferentes números de argumentos, quando a função é chamada, o Python, usará a definição que corresponde ao número de argumentos que são passados.

```
>>> myName = "Grétar"
```

```
>>> def sayHello(name):
```

```
        print("Hello",name)
        >>> sayHello(myName)
Hello Grétar
```

```
>>> def sayHello (name, country):
        print("Hello",name,"from",country)
        >>> sayHello(myName, myCountry)
Hello Grétar from Iceland
```

```
>>> myCountry = "Iceland"
```

Deve-se ter especial cuidado com variáveis no que diz respeito a funções. Neste tutorial aplicar-se-ão as seguintes regras às variáveis:
As variáveis criadas dentro de uma função só podem ser utilizadas dentro da função.
As variáveis criadas fora de uma função podem ser acedidas dentro da função, mas não se pode atribuir um novo valor. Tentar atribuir um novo valor dentro de uma função a uma variável criada fora da função criará uma nova variável com o mesmo nome dentro da função ou causará um erro.

```
>>> def insideVar():
        inVar = "inside"
        print(inVar)

>>> insideVar()
inside
>>> print(inVar)
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    print(inVar)
NameError: name 'inVar' is not defined
```

```
>>> outVar = "outside"
>>> def changeOutsideVar():
        outVar = "inside"
        print(outVar)

>>> changeOutsideVar()
inside
>>> print(outVar)
outside
```

Há maneiras de contornar estas regras, mas que está fora do âmbito deste tutorial

Entrada do utilizador

Para pedir a entrada de dados ao utilizador podemos utilizar a função "**input()**". Quando a função "**input()**" for chamada, o Python, aguarde até que o utilizador responda.

```
>>> def askForName():
        name = input("Enter your name: ")
        print("Hello",name)
>>> askForName()
Enter your name: Grétar
Hello Grétar
```

Por predefinição, a função "**input()**" devolve uma string, pelo que, se quisermos que o utilizador envie um número, temos de converter o tipo da resposta.

```
>>> def askForNumber():
        returnedString = input("Enter number to be squared: ")
        returnedNumber = float(returnedString)
        print("Your number squared is",returnedNumber**2)

>>> askForNumber()
Enter number to be squared: 7
Your number squared is 49.0
```

Neste caso, converteu-se a resposta a um número de virgula flutuante, mas também poderia ter-se convertido num inteiro usando "**int()**" em vez de "**float()**". Note-se que se a resposta for incompatível com o tipo que se tenta converter, causará um erro.

A entrada do utilizador pode ser convertida imediatamente (sem primeiro ser armazenada)

```
>>> def askForAnotherNumber():
        returnedNumber = float(input("Enter number to be squared: "))
        print("Your number squared is",returnedNumber**2)
>>> askForAnotherNumber()
Enter number to be squared: 8
Your number squared is 64.0
```

ICELAND

Kynning

Þessar Python leiðbeiningar nota IDLE, innifalda forritunarumhverfið sem fylgir málinu og er hægt að sækja á www.python.org . Í forritunarmálinu Python er myllumerkið (#) notað til þess að setja inn athugasemdir, svo þegar Python les myllumerkið er restin af línunni hunsuð

Breytur

Í Python þarf ekki að búa til breytuna fyrst, heldur verður hún til um leið og henni er gefið gildi. Hún þarf heldur ekki að vera ákvörðuð týpa. Breytum eru gefin gildi með því að nota jafnaðarmerkið (=)

```
>>> myFirstVariable = 5        # Integer
>>> mySecondVariable = 3.14    # Float
>>> myThirdVariable = "abc"    # String
```

Það er hægt að búa til fleiri en eina breytu í sömu línu.

```
>>> x, y = 5,3                 # x = 5 and y = 3
```

Reikniaðgerðir

Helstu stærðfræðiaðgerðatákn eru innbyggð í Python forritunarmálið.

```
>>> # Create variables
>>> x, y = 5,3
>>> print("x =", x, "and y =",y)
x = 5 and y = 3
>>> # Addition
>>> z = x + y
>>> print("z =",z)
z = 8
>>> # Subtraction
>>> z = x-y
>>> print("z =",z)
z = 2
```

```
>>> # Multiplication
>>> z = x*y
>>> print("z =",z)
z = 15
>>> # Division
>>> z = x/y
>>> print("z =",z)
z = 1.6666666666666667
>>> # Exponentiation (Power)
>>> z = x**y
>>> print("z =",z)
z = 125
```

Stærðfræðileg föll eru venjulega ekki innbyggð í Python en hægt er að flytja inn „math“ skránna til þess að hægt sé að nota föllin. Þá getum við kallað í föllin með því að setja „math.“ Á undan.

```
>>> import math
>>> a = 169
>>> b = math.sqrt(a)
>>> print("The square root of",a,"is",b)
The square root of 169 is 13.0
```

Höfum í huga að í Python, eins og öðrum forritunarmálum, getur litið út fyrir að tugabrot séu ónákvæm. Þetta er vegna þess hvernig tugabrot eru táknuð. Til dæmis, ef ég gef breytu gildið 0.1 verður hún ekki geymd sem nákvæmlega 1/10. Þetta getur gefið niðurstöður sem líta út fyrir að vera ónákvæmar, en villurnar eru það smáar að þetta er aðallega pirrandi og við munum hunsa það í þessum leiðbeiningum.

```
>>> x=3
>>> y=0.1
>>> print(x*y)
0.30000000000000004
```

Samanburðar virkjar

Til þess að bera saman gildi notum við samanburðarvirkja.

Virki	Þýðing	Dæmi
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
>=	Greater than or equal	x >= y
<	Less than	x < y
<=	Less than or equal	x <= y

Skilyrðingar virkjar

Til þess að taka ákvörðun byggða á skilyrði getum við notað „if“ .

```
>>> x,y = 5,3
>>> if x > y:
      print(x, "is greater than", y)

5 is greater than 3
```

Tökum eftir að í Python þarf að draga inn línur . Eftir að við skrifum „if x > y:“, þurfa næstu skipanir að vera inndregnar svo þær séu lesnar þegar skilyrðin eru uppfyllt.
Til þess að fara út úr „if“ setningunni, ýttu þá tvisvar á return eftir síðustu skipun. Ef skipanirnar inni í „if“ setningunni eru ekki inndregnar skilar forritið villur.

```
>>> if x > y:
print(x, "is greater than", y)
SyntaxError: expected an indented block
```

Ef við viljum gera eitthvað annað þegar skilyrðið er ekki uppfyllt notum við „else“ skipunina.

```
>>> x, y = 5,3
>>> if y > x:
      print(y, "is greater than", x)
else:
      print(y, "is not greater than", x)
3 is not greater than 5
```

Tökum eftir að það þarf líka að draga inn kóðalínurnar undir „else“ skipuninni. Eftir að við höfum skrifað þann kóða sem við viljum að sé lesinn undir „else“ skipuninni, ýtum við á backspace.

Við getum líka notað setninguna „elif“ fyrir aukin skilyrði.

```
>>> x, y = 4,4
>>> if x > y:
      print(x, "is greater than", y)
elif y > x:
      print(y, "is greater than", x)
else:
      print(x, "is equal to", y)

      4 is equal to 4
```

Lykkjur og raðir

Til þess að endurtaka sömu skipanirnar getum við notað „for“ setninguna. Hún er lykkja sem endurtekur sig eftir röð og notar ítrarnir til þess. Við geymum stöðuna á röðinni í ítrunarbreytunni. „for“ lykkjan er eftirfarandi: for [iterator] in [sequence]:

Það er gott að vita um „range()“ þegar maður býr til „for“ lykkjur. „range()“ fallið býr til raðir af tölum, range(m) býr til röðina frá 0 til m-1.

```
>>> for i in range(3):
      print(i)

0
1
2
```

Eins og í „if“ setningunni þarf að draga inn þær línur sem á að endurtaka í „for lykkjunni“ og til þess að loka henni ýtum við tvisvar á return eftir síðustu skipun.

„range()“ fallið getur tekið inn byrjunargildi, range(n, m) býr til röð af tölum frá n til m-1. Það er líka hægt að gefa fallinu mismunandi aukningarbil, range(n, m, q) býr til röð af tölum frá n til m-1 og hækkar sig um q á milli talna. Öll gildi sem fara í gegnum „range()“ fallið þurfa að vera heiltölur.

Einnig er hægt að búa til röðina áður en „for“ lykkjan hefst.

```
>>> myFirstSequence = range(2,5)
>>> for i in myFirstSequence:
      print(i, "is in sequence")

2 is in sequence
3 is in sequence
4 is in sequence
```

```
>>> for i in mySecondSequence:
      print(i, "is in sequence")


4 is in sequence
6 is in sequence
8 is in sequence
```

```
>>> mySecondSequence = range(4,10,2)
```

Aðrar týpur af röðum fyrir „for“ lykkjuna geta verið strengir eða listar.

```
>>> myString = "GBPE"
>>> for i in myString:
      print(i,"is in 'GBPE' ")

G is in 'GBPE'
B is in 'GBPE'
P is in 'GBPE'
E is in 'GBPE'
```

```
>>> myList = ["CZ","IS","IT","PT","ES"]
>>> for i in myList:
      print(i,"is in GBPE")

CZ is in GBPE
IS is in GBPE
IT is in GBPE
PT is in GBPE
ES is in GBPE
```

Föll

Til þess að skilgreina föll í Python notum við „def“ orðið. Eins og áður þarf að draga inn línur innan fallsins og ýta tvisvar á return til þess að loka því.

```
>>> def sayHello():
      print("Hello world!")


>>> sayHello()
Hello world!
```

Co-funded by the
Erasmus+ Programme
of the European Union

Sviginn aftan við nafn fallsins er nauðsynlegur bæði þegar við skilgreinum fallið og þegar við köllum í það. Ef við viljum að fallið taki inn breytur þá fara þær í svigann. Það er hægt að skilgreina sama fallið fyrir mismunandi fjölda breyta og Python mun nota þá skilgreiningu sem passar við fjölda breyta í sviganum.

```
>>> myName = "Grétar"
>>> def sayHello(name):
      print("Hello",name)


>>> sayHello(myName)
Hello Grétar
```

```
>>> myCountry = "Iceland"
>>> def sayHello (name, country):
      print("Hello",name,"from",country)


>>> sayHello(myName, myCountry)
Hello Grétar from Iceland
```

Það þarf að passa sig með breytur þegar kemur að föllum. Í þessu tilviki munu eftirfarandi reglur eiga við um breytur:

Breytur sem búnar eru til inni í falli er einungis hægt að nota inni í fallinu.

Breytur sem búnar eru til fyrir utan fall er hægt að kalla í inni í fallinu en ekki hægt að gefa þeim nýtt gildi. Ef reynt er að gefa breytu sem búin er til utan falls nýtt gildi verður annaðhvort til ný breyta með sama nafni eða að kóðinn skilar villu.

```
>>> def insideVar():
      inVar = "inside"
      print(inVar)

>>> insideVar()
inside
>>> print(inVar)
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in
<module>
    print(inVar)
NameError: name 'inVar' is not
defined
```

```
>>> outVar = "outside"
>>> def changeOutsideVar():
      outVar = "inside"
      print(outVar)


>>> changeOutsideVar()
inside
>>> print(outVar)
outside
```

Það er hægt að komast framhjá þessu, en við förum ekki í gegnum það í þessum leiðbeiningum.

Notendaílag

Til þess að biðja notanda forrits um að skrifa eitthvað notum við „input()“ fallið. Þegar kallað er í fallið bíður Python eftir að notandi skrifi svarið.

```
>>> def askForName():
      name = input("Enter your name: ")
      print("Hello",name)


>>> askForName()
Enter your name: Grétar
Hello Grétar
```

„Input()“ fallið skilar streng svo ef við viljum fá tölu verðum við að breyta svarinu.

```
>>> def askForNumber():
      returnedString = input("Enter number to be squared: ")
      returnedNumber = float(returnedString)
      print("Your number squared is",returnedNumber**2)


>>> askForNumber()
Enter number to be squared: 7
Your number squared is 49.0
```

Í þessu tilviki breytti ég svarinu í float tölu en ég hefði líka getað breytt henni í heiltölu með því að nota „int()“ í stað „float()“. Ef svar notandans passar ekki við týpuna sem þú reynir að breyta í þá skilar forritið villu.

Það er hægt að umbreyta svarinu án þess að geyma það fyrst í breytu.

```
>>> def askForAnotherNumber():
      returnedNumber = float(input("Enter number to be squared: "))
      print("Your number squared is",returnedNumber**2)


>>> askForAnotherNumber()
Enter number to be squared: 8
Your number squared is 64.0
```

ITALIA

Introduzione

Questo tutorial Python utilizza IDLE, l'ambiente di sviluppo integrato che viene fornito in bundle con l'implementazione predefinita del linguaggio che può essere scaricato da www.python.org.
In Python il carattere hash (#) viene utilizzato per inserire commenti, quando Python incontra un carattere hash ignorerà il resto della riga.
Variabili
In Python una variabile non ha bisogno di essere dichiarata, viene creata la prima volta che le viene assegnato un valore. Inoltre, non è necessario crearlo come un tipo particolare. I valori vengono assegnati alle variabili utilizzando il segno di uguale (=).

```
>>> myFirstVariable = 5        # Integer
>>> mySecondVariable = 3.14    # Float
>>> myThirdVariable = "abc"    # String
```

È possibile creare più variabili in una riga.

```
>>> x, y = 5,3                 # x = 5 and y = 3
```

Operatori aritmetici

Gli operatori matematici di base sono integrati nel linguaggio di programmazione Python.

```
>>> # Create variables
>>> x, y = 5,3
>>> print("x =", x, "and y =",y)
x = 5 and y = 3
>>> # Addition
>>> z = x + y
>>> print("z =",z)
z = 8
>>> # Subtraction
>>> z = x-y
>>> print("z =",z)
z = 2
```

```
>>> # Multiplication
>>> z = x*y
>>> print("z =",z)
z = 15
>>> # Division
>>> z = x/y
>>> print("z =",z)
z = 1.6666666666666667
>>> # Exponentiation (Power)
>>> z = x**y
>>> print("z =",z)
z = 125
```

Le funzioni matematiche generalmente non sono integrate in Python, quindi per usarle dobbiamo prima importare il modulo "math". Chiamiamo quindi le funzioni con il prefisso "math".

```
>>> import math
>>> a = 169
>>> b = math.sqrt(a)
>>> print("The square root of",a,"is",b)
The square root of 169 is 13.0
```

Va notato che in Python, come altri linguaggi di programmazione, l'aritmetica in virgola mobile può sembrare imprecisa. Ciò è dovuto al modo in cui sono rappresentate le variabili in virgola mobile. Ad esempio, se assegno il valore 0.1 a una variabile, in effetti non verrà memorizzata esattamente come 1/10. Questo può produrre risultati che sembrano imprecisi ma gli errori sono così piccoli che è per lo più solo fastidioso e lo ignoreremo in questo tutorial.

```
>>> x=3
>>> y=0.1
>>> print(x*y)
0.30000000000000004
```

Operatori di confronto

Per confrontare i valori utilizziamo gli operatori di confronto.

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

Dichiarazioni condizionali

Per prendere decisioni in base alle condizioni possiamo utilizzare le affermazioni "if".

```
>>> x,y = 5,3
>>> if x > y:
      print(x, "is greater than", y)
5 is greater than 3
```

Nota che Python usa il rientro per raggruppare le istruzioni. Dopo aver digitato "if x > y:", le seguenti righe di comando che devono essere eseguite se la condizione è vera sono rientrate. Per uscire dall'istruzione "if", premere Invio due volte dopo l'ultimo comando. Non avere le righe di comando all'interno dell'istruzione "if" indentata causerà un errore.

```
>>> if x > y:
print(x, "is greater than", y)
SyntaxError: expected an indented block
```

Se vogliamo fare qualcos'altro se la condizione non è soddisfatta utilizziamo la parola chiave “else”.

```
>>> x, y = 5,3
>>> if y > x:
      print(y, "is greater than", x)
else:
      print(y, "is not greater than", x)
3 is not greater than 5
```

Notare ancora l'indentazione, la parola chiave "else" deve avere la stessa indentazione della parola chiave "if" a cui è associata. Dopo aver inserito l'ultimo comando da eseguire se l'istruzione è soddisfatta, premere backspace.
Possiamo anche avere condizioni aggiuntive usando la parola chiave "elif".

```
>>> x, y = 4,4
>>> if x > y:
      print(x, "is greater than", y)
elif y > x:
      print(y, "is greater than", x)
else:
      print(x, "is equal to", y)

      4 is equal to 4
```

Co-funded by the
Erasmus+ Programme
of the European Union

Cicli e Sequenze

Per ripetere lo stesso insieme di comandi possiamo usare la parola chiave “for”. Il ciclo "for" esegue un'iterazione su una sequenza e utilizza un iteratore (variabile) per memorizzare il valore corrente dalla sequenza. La sintassi del ciclo "for" è: for [iterator] in [sequence]:
È utile conoscere la funzione "range()" durante la creazione di loop "for". La funzione "range()" crea una sequenza di numeri, range(m) crea una sequenza di m numeri da 0 a m-1.

```
>>> for i in range(3):
      print(i)

0
1
2
```

Come nelle istruzioni "if", l'indentazione viene utilizzata per raggruppare le istruzioni e per chiudere il ciclo "for" premiamo invio due volte dopo l'ultimo comando.
Alla funzione "range()" può essere assegnato un valore iniziale diverso, range(n, m) crea una sequenza di numeri da n a m-1. Può anche essere assegnato un valore di incremento diverso, range(n, m, q) crea una sequenza di numeri da n a m-1 incrementando di q tra i numeri. Tutti i valori passati alla funzione "range()" devono essere interi.
La sequenza su cui ripetere l'iterazione può anche essere creata prima del ciclo "for".

```
>>> myFirstSequence = range(2,5)
>>> for i in myFirstSequence:
      print(i, "is in sequence")

2 is in sequence
3 is in sequence
4 is in sequence
```

```
>>> for i in mySecondSequence:
      print(i, "is in sequence")


4 is in sequence
6 is in sequence
8 is in sequence
```

```
>>> mySecondSequence = range(4,10,2)
```

Altri tipi di sequenze su cui il ciclo "for" può scorrere includono stringhe ed elenchi.

```
>>> myString = "GBPE"
>>> for i in myString:
      print(i,"is in 'GBPE' ")

G is in 'GBPE'
B is in 'GBPE'
P is in 'GBPE'
E is in 'GBPE'
```

```
>>> myList = ["CZ","IS","IT","PT","ES"]
>>> for i in myList:
      print(i,"is in GBPE")

CZ is in GBPE
IS is in GBPE
IT is in GBPE
PT is in GBPE
ES is in GBPE
```

Funzioni

Per definire le funzioni in Python utilizziamo la parola chiave “def”. Come prima, Python usa l'indentazione per raggruppare i comandi e per completare la definizione della funzione premere Invio due volte dopo l'ultimo comando.

```
>>> def sayHello():
      print("Hello world!")


>>> sayHello()
Hello world!
```

La parentesi dopo il nome della funzione è necessaria sia quando si definisce la funzione che quando la si chiama. Se vogliamo passare argomenti alla funzione lo indicheremmo tra parentesi. È possibile definire la stessa funzione per diversi numeri di argomenti, quando la funzione viene chiamata Python utilizzerà la definizione che corrisponde al numero di argomenti passati.

```
>>> myName = "Grétar"
>>> def sayHello(name):
      print("Hello",name)


>>> sayHello(myName)
Hello Grétar
```

```
>>> myCountry = "Iceland"
>>> def sayHello (name, country):
      print("Hello",name,"from",country)


>>> sayHello(myName, myCountry)
Hello Grétar from Iceland
```

È necessario prestare particolare attenzione alle variabili quando si tratta di funzioni. In questo tutorial le seguenti regole si applicheranno alle variabili:
Le variabili create all'interno di una funzione possono essere utilizzate solo all'interno della funzione. È possibile accedere alle variabili create all'esterno di una funzione all'interno della funzione ma non a un nuovo valore. Il tentativo di assegnare un nuovo valore all'interno di una funzione a una variabile creata all'esterno della funzione creerà una nuova variabile con lo stesso nome all'interno della funzione o causerà un errore.

```
>>> def insideVar():
      inVar = "inside"
      print(inVar)

>>> insideVar()
inside
>>> print(inVar)
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    print(inVar)
NameError: name 'inVar' is not defined
```

```
>>> outVar = "outside"
>>> def changeOutsideVar():
      outVar = "inside"
      print(outVar)


>>> changeOutsideVar()
inside
>>> print(outVar)
outside
```

Ci sono modi per aggirare queste regole, ma questo va oltre lo scopo di questo tutorial

Imput utente

Per chiedere input all'utente possiamo usare la funzione “input()”. Quando viene chiamata la funzione "input()", Python attenderà finché l'utente non risponde.

```
>>> def askForName():
      name = input("Enter your name: ")
      print("Hello",name)


>>> askForName()
Enter your name: Grétar
Hello Grétar
```

Di default la funzione “input()” restituisce una stringa quindi se vogliamo che l'utente invii un numero dobbiamo convertire il tipo di risposta.

```
>>> def askForNumber():
      returnedString = input("Enter number to be squared: ")
      returnedNumber = float(returnedString)
      print("Your number squared is",returnedNumber**2)


>>> askForNumber()
Enter number to be squared: 7
Your number squared is 49.0
```

In questo caso ho convertito la risposta in un numero in virgola mobile, ma avrei anche potuto convertirla in un numero intero usando "int()" invece di "float()". Nota che se la risposta è incompatibile con il tipo in cui provi a convertirla causerà un errore.
L'input dell'utente può essere convertito immediatamente (senza essere prima memorizzato)

```
>>> def askForAnotherNumber():
      returnedNumber = float(input("Enter number to be squared: "))
      print("Your number squared is",returnedNumber**2)


>>> askForAnotherNumber()
Enter number to be squared: 8
Your number squared is 64.0
```

SPAIN

Introducción

Este tutorial de Python utiliza IDLE, el entorno de desarrollo integrado que viene incluido con la implementación predeterminada del lenguaje que se puede descargar desde www.python.org.

En Python, el carácter hash (#) se usa para insertar comentarios, cuando Python encuentra un carácter hash, ignorará el resto de la línea.

Variables

En Python no es necesario declarar una variable, se crea la primera vez que se le asigna un valor. Tampoco es necesario crearlo como un tipo particular. Los valores se asignan a las variables utilizando el signo igual (=).

```
>>> myFirstVariable = 5          # Integer
>>> mySecondVariable = 3.14      # Float
>>> myThirdVariable = "abc"      # String
```

Se pueden crear múltiples variables en una línea.

```
>>> x, y = 5,3                   # x = 5 and y = 3
```

Operadores aritméticos

Los operadores matemáticos básicos están integrados en el lenguaje de programación Python

```
>>> # Create variables
>>> x, y = 5,3
>>> print("x =", x, "and y =",y)
x = 5 and y = 3
>>> # Addition
>>> z = x + y
>>> print("z =",z)
z = 8
>>> # Subtraction
>>> z = x-y
>>> print("z =",z)
z = 2
```

```
>>> # Multiplication
>>> z = x*y
>>> print("z =",z)
z = 15
>>> # Division
>>> z = x/y
>>> print("z =",z)
z = 1.6666666666666667
>>> # Exponentiation (Power)
>>> z = x**y
>>> print("z =",z)
z = 125
```

Las funciones matemáticas generalmente no están integradas en Python, por lo que para usarlas primero debemos importar el módulo "matemáticas". Luego llamamos a las funciones con el prefijo "matemáticas".

```
>>> import math
>>> a = 169
>>> b = math.sqrt(a)
>>> print("The square root of",a,"is",b)
The square root of 169 is 13.0
```

Cabe señalar que en Python, al igual que otros lenguajes de programación, la aritmética de coma flotante puede parecer inexacta. Esto se debe a la forma en que se representan las variables de punto flotante. Por ejemplo, si asigno el valor 0.1 a una variable, de hecho no se almacenará

exactamente como 1/10. Esto puede producir resultados que parecen inexactos, pero los errores son tan pequeños que en su mayoría son molestos y los ignoraremos en este tutorial.

```
>>> x=3
>>> y=0.1
>>> print(x*y)
0.30000000000000004
```

Operadores de comparación

Para comparar valores usamos operadores de comparación.

Operator	Meaning	Ejemplo
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
>=	Greater than or equal	x >= y
<	Less than	x < y
<=	Less than or equal	x <= y

Declaraciones condicionales

Para tomar decisiones basadas en condiciones podemos usar las declaraciones „si“.

```
>>> x,y = 5,3
>>> if x > y:
        print(x, "is greater than", y)

5 is greater than 3
```

Tenga en cuenta que Python usa sangría para agrupar declaraciones. Después de escribir „if x > y:“, las siguientes líneas de comando que se ejecutarán si la condición es verdadera están sangradas. Para salir de la declaración "if", presione regresar dos veces después del último comando. No tener las líneas de comando dentro de la instrucción "si" con sangría provocará un error.

```
>>> if x > y:
print(x, "is greater than", y)
SyntaxError: expected an indented block
```

Si queremos hacer otra cosa si no se cumple la condición, usamos la palabra clave "else".

```
>>> x, y = 5,3
>>> if y > x:
        print(y, "is greater than", x)
else:
        print(y, "is not greater than", x)
3 is not greater than 5
```

Tenga en cuenta nuevamente la sangría, la palabra clave "else" debe tener la misma sangría que la palabra clave "si" con la que está asociada. Después de ingresar el último comando que se ejecutará si se cumple la declaración, presione la tecla de retroceso.
También podemos tener condiciones adicionales usando la palabra clave “elif”. .

```
>>> x, y = 4,4
>>> if x > y:
        print(x, "is greater than", y)
elif y > x:
        print(y, "is greater than", x)
else:
        print(x, "is equal to", y)
4 is equal to 4
```

Bucles y secuencias

Para repetir el mismo conjunto de comandos, podemos usar la palabra clave "for". El ciclo "for" itera sobre una secuencia y usa un iterador (variable) para almacenar el valor actual de la secuencia. La sintaxis del bucle "for" es: para [iterador] en [secuencia]:

Es útil conocer la función "rango ()" al crear bucles "for". La función “rango()” crea una secuencia de números, range(m) crea una secuencia de m números de 0 a m-1.

```
>>> for i in range(3):
        print(i)
0
1
2
```

Al igual que en las declaraciones "si", la sangría se usa para agrupar declaraciones y para cerrar el ciclo "for" presionamos regresar dos veces después del último comando.

A la función “rango()” se le puede dar un valor inicial diferente, rango(n, m) crea una secuencia de números de n a m-1. También se le puede dar un valor de incremento diferente, range(n, m, q) crea una secuencia de números de n a m-1 incrementando por q entre números. Todos los valores pasados a la función “rango()” deben ser números enteros.

La secuencia que se repetirá también se puede crear antes del bucle "for".

```
>>> myFirstSequence = range(2,5)
>>> for i in myFirstSequence:
        print(i, "is in sequence")

2 is in sequence
3 is in sequence
4 is in sequence
>>> mySecondSequence = range(4,10,2)
```

```
>>> for i in mySecondSequence:
        print(i, "is in sequence")


4 is in sequence
6 is in sequence
8 is in sequence
```

Other types of sequences the “for” loop can iterate over include strings and lists.

```
>>> myString = "GBPE"
>>> for i in myString:
        print(i,"is in 'GBPE' ")

G is in 'GBPE'
B is in 'GBPE'
P is in 'GBPE'
E is in 'GBPE'
```

```
>>> myList = ["CZ","IS","IT","PT","ES"]
>>> for i in myList:
        print(i,"is in GBPE")

CZ is in GBPE
IS is in GBPE
IT is in GBPE
PT is in GBPE
ES is in GBPE
```

Las funciones matemáticas generalmente no están integradas en Python, por lo que para usarlas.