Python Tutorial

References

- https://www.w3schools.com/python/
- Python cơ bản
- A Practical Introduction to Python Programming

Introduction

- Python is a popular programming language. It was created by Guido van Rossum, and released in 1991.
- It is used for:
  - web development (server-side),
  - software development,
  - mathematics,
  - system scripting.

Comments

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.
- One line comments:

```
# This is a comment
```

- Multi-line comments:

```
""" This is a multi-line comment
written in more than just one line"""
```

Variables

- Variables are containers for storing data values.
- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)
```


Variables

- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- A variable name must start with a letter or the underscore character
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Legal variable names: myvar, my_var, _my_var, myVar, myvar2

Illegal variable names: 2myvar, my-var, my var

Variables

- Assign value to multiple variables:

```
x, y, z = "Orange", "Banana", "Cherry“
x = y = z = "Orange“
```

- Output variables:

```
name = 'Tran Thanh Hung'
n = 123456789
x = 123456789.98765
print('Name: ', name)
print('Name: ' + name)
print(f'Name: {name}')
print(f'Variables: n = {n} and x = {x}')
print(f'Variables: n = {n:,} and x = {x:,.2f}')
```


Data Types

- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:
- Getting the data type:

```
x = 5
print(type(x)) # int
```

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

Data Types

- The data type is set when you assign a value to a variable:

Example	Data Type
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "John", "age" : 36}	dict
x = {"apple", "banana", "cherry"}	set
x = True	bool

Data Types

- You can specify the data type:

Example	Data Type
x = str("Hello World")	str
x = int(20)	int
x = float(20.5)	float
x = complex(1j)	complex
x = list(("apple", "banana", "cherry"))	list
x = tuple(("apple", "banana", "cherry"))	tuple
x = range(6)	range
x = dict(name="John", age=36)	dict
x = set(("apple", "banana", "cherry"))	set

Numbers

- There are three numeric types in Python:
  - int
  - float
  - Complex
- Example:

```
x = 1        # int
y = 2.8      # float
z = 1j       # complex
```


Strings

- String Literals
  - String literals in python are surrounded by either single quotation marks, or double quotation marks.

    'hello' is the same as "hello"
- Multiline Strings
  - You can assign a multiline string to a variable by using three quotes:

```
a = '''Thời gian là miễn phí nhưng nó vô giá.
Bạn không thể sở hữu nó, nhưng bạn có thể sử dụng nó.
Bạn có thể dùng nó, nhưng bạn không thể giữ nó.
Một khi bạn làm mất nó, bạn sẽ không thể nào có lại được nó.'''
print(a)
```

Strings are Arrays

- Get the character at position 1 (the first character has the position 0):

```
a = "Hello, World!"
print(a[1]) # e
```

- Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[1:5]) # ello
```

- To get the length of a string, use the len() function.

```
a = "Hello World!"
print(len(a)) # 12
```


Arithmetic Operators

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

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3

Comparison Operators

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

Logical Operators

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

Collections (Arrays)

- There are four collection data types in the Python programming language:
  - **List** is a collection which is ordered and changeable. Allows duplicate members.
  - **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
  - **Set** is a collection which is unordered and unindexed. No duplicate members.
  - **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

List

- A list is a collection which is ordered and changeable.
- Create a List:

```
list = ["apple", "banana", "cherry"]
print(list)
```

- You access the list items by referring to the index number:

```
print(list[1]) # banana
```

- Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second last item etc.
- Print the last item of the list:

```
print(thislist[-1]) # cherry
```


List

- Returns the third, fourth, and fifth item:

```
list = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(list[2:5]) # ['cherry', 'orange', 'kiwi']
```

- Returns the items from the beginning to "orange":

```
print(list[:4]) # ['apple', 'banana', 'cherry', 'orange']
```

- Returns the items from "cherry" to the end:

```
print(list[2:]) # ['cherry', 'orange', 'kiwi', 'melon', 'mango']
```

List

- Print all items in the list, one by one:

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

- To determine how many items a list has, use the len() function:

```
print(len(thislist))
```


If … Else

- If statement:

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

- Elif:

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

- Else:

```
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")
```

If ... Else

- And:

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

- Or:

```
a = 200
b = 33
c = 500
if a > b or a > c:
    print("At least one of the conditions is True")
```


While Loops

- The while loop executes a set of statements as long as a condition is true.
- Print numbers from 1 to 5:

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

For Loops

- A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).
- Print each item in a list:

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

- Looping through a string:

```
for x in "banana":
    print(x)
```


Functions

- A function is a 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.
- Creating a function:

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

- Calling a function:

```
my_function()
```

Functions

- Information can be passed into functions as arguments or parameters.
- Creating a function with arguments:

```
def add(a, b):
    return a+b
print(add(2,3)) # 5
```


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

- A lambda function that adds 10 to the argument:

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

- It is similar to: →

```
def x(a):
    return a + 10
print(x(5))
```

Thank you!

QUESTIONS?