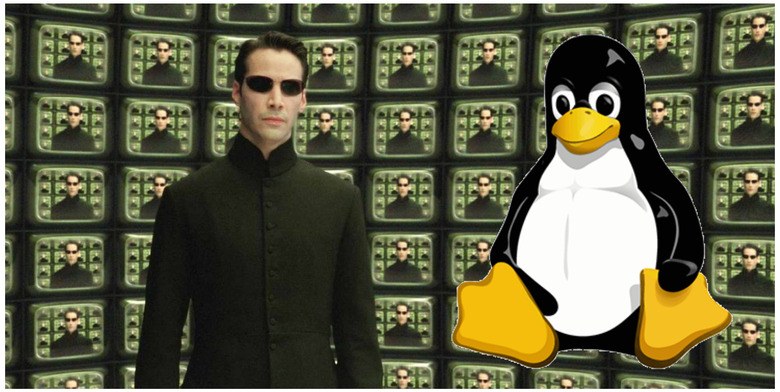
17.09.21 14:50

CSCI 120
Introduction to Computer Science and Programming I
Lecture 1: Your First Python Programme III



Jetic Gū

Overview

- Focus: Basic C/C++ Syntax
- Architecture: Linux/Unix OS
- Core Ideas:
    1. More on `str`
    2. `if` conditions

More on `str`

P1
str

Variables

- `int` variables
  - Integers, converts to `float` automatically when seeing fractions
  - can be converted to `str` using `str(var)`
- `float` variables
  - Fractions, can be converted to `str` using `str(var)`
- `str` variables
  - Valid variables can be converted to `int` and `float` using `int(var)` and `float(var)`

Review

P1
Variables

What is happening in `str`?

```
var = "Hello World!"
```

- `str` variable
  - a string of characters
  - each character is an element in the string

Concept

P1
Variables

What is happening in `str`?

```
var = "Hello World!"
```

Memory Table

Address	Variable	Type	Value
Slot 0	var	`str`	`"Hello World!"`
....			

What is happening in `str`?

```
var = "Hello World!"
print(var[0])
```

- Individual characters can be accessed by index
- Index can be specified using square brackets

Memory Table

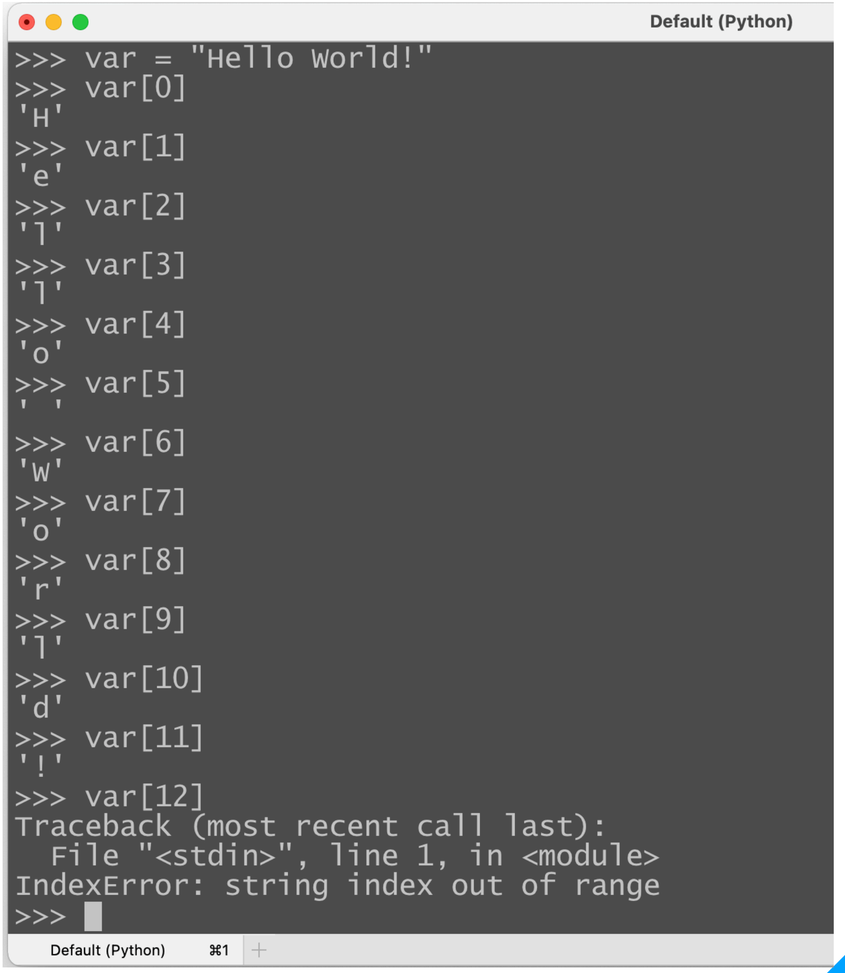
Address	Variable	Type	Value
Slot 0	var	str	"H"
			"e"
			"l"
			"l"
			"o"
			" "
....			...

P1
Variables

What is happening in `str`?

```
var = "Hello World!"
print(var[0])
```

- Individual characters can be accessed by index
- Index can be specified using square brackets



Technical

P1
Variables

What is happening in `str`?

```
var = "Hello World!"
print(var[0])
```

- You can get the length of a string using `len(var)`
- You can get the last character of a string using `var[-1]`

Technical

Advanced Indexing

- Python supports complex indexing with ranges: `var[i:j]`, meaning from index `i` to `j-1`
- e.g. substring for `var = "Hello World!"`
  - `print(var[6:9])`
    This will return a substring with `var[6], var[7], var[8]: "Wor"`
  - `print(var[:5])`
    This will return the first 5 characters as a substring: `"Hello"`
  - `print(var[6:])`
    This will return the entire substring starting from `var[6]`: `"World!"`
  - `print(var[6:-1])`
    This will return the entire substring starting from `var[6]`, until the 2nd last: `"World"`

Concept

`str` splitting method

- A string can be split into a list of substrings, separated by whitespaces (space, tabs, newline symbols, etc.)

```
aList = "I  am\ta\nsentence".split()
# This is called a method of string class
```

  - `print(aList)`
    This prints `['I', 'am', 'a', 'sentence']`

  - `print(aList[3])`
    Lists can be accessed by index, this prints `sentence`

- We'll talk more about `list`s next week once we've covered loops

Concept

Conditions

`if` conditions, logical expressions, basic subroutine

P2
Conditions

Conditional Statements

```
if EXPRESSIONS:
    DO STUFF
```

- Conditional expressions allow you to execute code only when certain conditions are met
    - These conditions are mathematical expressions that can be converted to boolean values (`bool`)

Concept

Boolean Values

- `bool` class
  - A special type that has only two values: `True` and `False`
- `bool` values can be generated as results of comparisons or assigned to variables
  - Arithmetic comparison operators:
    - Equal, not equal: `==`, `!=`
    - Greater/Less than: `>`, `<`
    - Greater/Less than or equal to: `>=`, `<=`

```
>>> 34 > 29
True
>>> 56 < 12
False
>>> 55 >= 55
True
>>> 53 <= 38
False
```

```
>>> 1234 == 1234
True
>>> 1234 == 7890
False
>>> 1234 != 1234
False
>>> 1234 != 7890
True
>>>
```

Examples

Conditional Statements

```
if var1 > var2:
    print("var1 is greater than var2")
```

A subroutine

- The expression in `if` statement must be (or be convertible to) a boolean value
  - Execution: subroutine

Concept

Python subroutines

```
if var1 > var2:
    print("1st print, inside if subroutine")
    print("2nd print, inside if subroutine")
    print("3rd print, inside if subroutine")
print("last print, outside if subroutine")
```

- A subroutine is a block of code that is executed together
- Python subroutine is specified by **indentation, usually using 4 space bars**
- `if` condition requires code to be inside a subroutine

Concept

Python subroutines

```
var1 = input()
var2 = input()

if var1 == "Missile":
    print("Confirming missile strike")
    if var2 == "Confirm":
        print("Missile strike confirmed")

if var1 == "Sleep":
    print("Confirming sleep order")
    if var2 == "Bed":
        print("Sleep inside my bed")
```

A subroutine

A subroutine

A subroutine

A subroutine

Python subroutines

- Maintain consistent indentation
  - Recommended:
    use increment of 4 space bars
  - Do not mix tabs (`\t`) with space bars

```
if var1 > var2:
print("...")
print("...")
# doesn't work, no subroutine

if var1 > var2:
 print("...")
  print("...")
# doesn't work

if var1 > var2:
    print("...")   # 4 space bar
    print("...")   # 1 tab '\t'
# doesn't work
```

Concept

`else` **and** `elif`

```
if var1 > var2:
    print("var1 is greater than var2")
else:
    print("var1 is not greater than var2")
```

- `else` must be on the same routine level as `if`
  - `else` subroutine will be executed when `if` condition is not met

Concept

`else` **and** `elif`

```
if var1 > var2:
    print("var1 is greater than var2")
elif var1 == var2:
    print("var1 is equal to var2")
else:
    print("var1 is less than var2")
```

- `elif` is like else, but with another condition
  - same routine level as `if`
  - `elif` condition will be examined when conditions above are not met
  - can have multiple `elif` under a single `if`

Concept

else **and** elif

```
var = input()
if var == "1":
    print("Subroutine 1")
elif var == "2":
    print("Subroutine 2")
elif var == "3":
    print("Subroutine 3")
elif var == "4":
    print("Subroutine 4")
else:
    print("Subroutine 5")
```

Example

`bool` expressions

```
if var1 == "Missile Launch" and var2 == "Confirm":
    print("Missile launch confirm")

if (person1 == "Yes" and person2 == "Yes") or (president == "Yes"):
    print("Authorised")
```

- You can combine multiple `bool` values to form complex conditions using `and`, `or`, `not`, and parentheses
  - `bool1 and bool2`
    returns `True` when `bool1` and `bool2` are both `True`, otherwise `False`
  - `bool1 or bool2`
    returns `True` when either `bool1` or `bool2` is `True`, otherwise `False`
  - `not bool1`
    returns `opposite value`

Example