내가 그리스도와 함께 십자가에 못 박혔나니 그런즉 이제는 내가 사는 것이 아니요 오직 내 안에 그리스도께서 사시는 것이라 이제 내가 육체 가운데 사는 것은 나를 사랑하사 나를 위하여 자기 자신을 버리신 하나님의 아들을 믿는 믿음 안에서 사는 것이라 (갈2:20)

Python for Machine Learning

Lecture Notes by idebtor@gmail.com, Handong Global University

NOTE: The following materials have been compiled and adapted from the numerous sources including my own. Please help me to keep this tutorial up-to-date by reporting any issues or questions. Send any comments or criticisms to `idebtor@gmail.com` Your assistances and comments will be appreciated.

Revisit cell magic commands

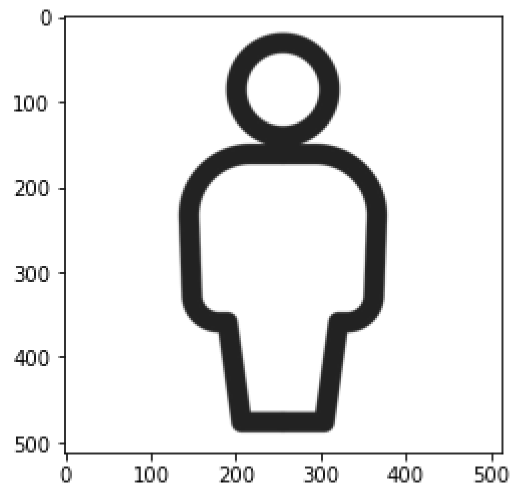
Exercise 1.

1. Save the following cell in `person.py` .
2. Run the code at the console (terminal): python person.

3. Read `person.py` file into a cell.

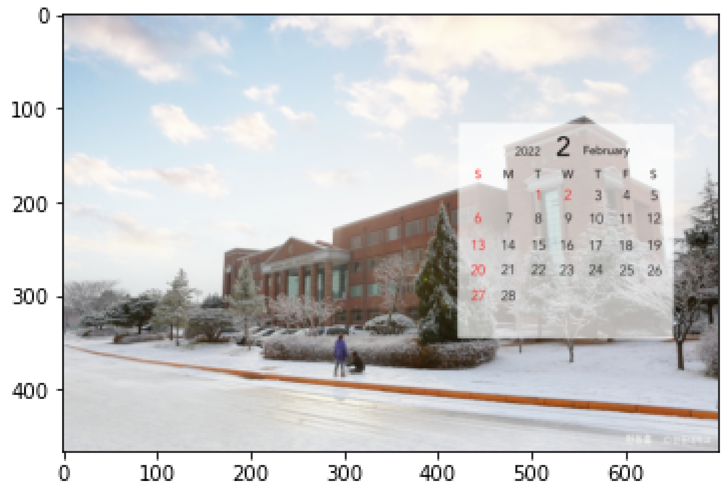
hint: %%writefile, %load

In [1]:
```
%%writefile person.py
# this is a way to read a file from a url
import numpy as np
import matplotlib.pyplot as plt
import urllib.request
import PIL

url = 'https://github.com/idebtor/KMOOC-ML/blob/master/ipynb/images/joyai/person.png?
img = np.array(PIL.Image.open(urllib.request.urlopen(url)))
plt.imshow(img)
plt.show()
```



In [2]:
```
%%writefile handong.py
# this is a way to read a file from a local drive

import matplotlib.pyplot as plt
```

```python
from matplotlib.image import imread
img = imread('./images/2202.png')
plt.imshow(img)
plt.show()
```



a solution:

Exercise 2: Run the python scripts at the console.

1. Start Python with `person.py` script at the console(or terminal).
    - `python`
2. Start Python and `import person` module at the >>> prompt.
    - `python`

Data Types and Operations

학습 목표

- 데이터 타입에 대해 이해한다.
- 간단한 연산들을 수행할 수 있다.

학습 내용

1. 파이썬 데이터 타입 살펴보기
2. 데이터 타입별 사용방법 익히기

- In programming, data type is an important concept.
- 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:

> Text Type: `str`
>
> Numeric Types: `int, float, complex`
>
> Sequence Types: `list, tuple, range`
>
> Mapping Type: `dict`
>
> Set Types: `set, frozenset`
>
> Boolean Type: `bool`

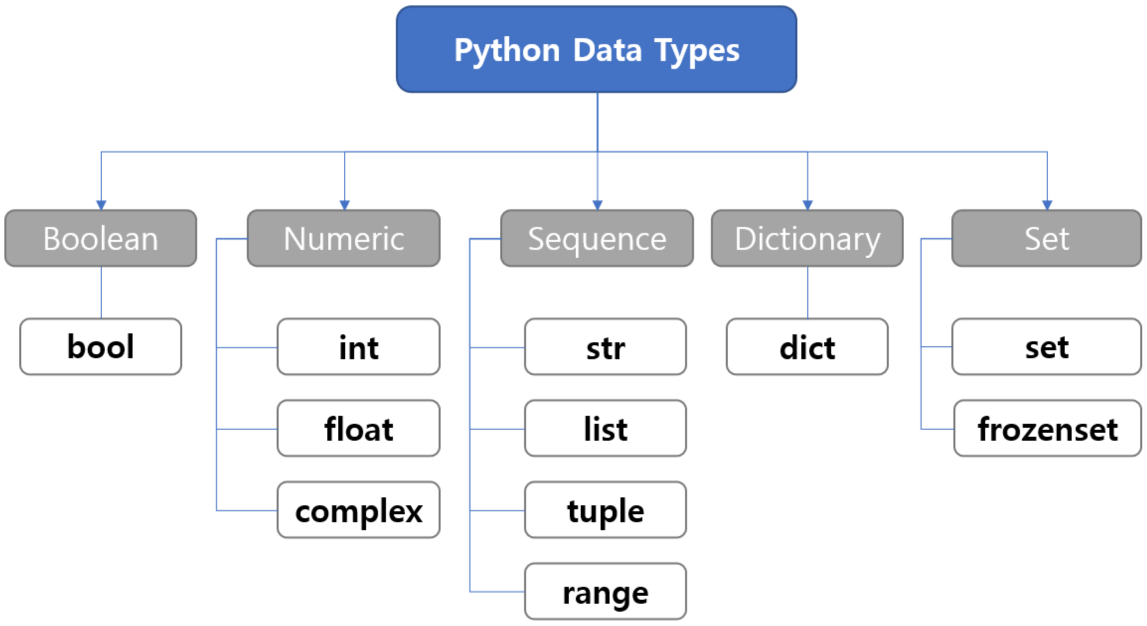


Figure 1. Built-in Data Types in Python

Setting the data type

The data type in Python is set **when you assign a value to a variable:**

Let's take a look at the common numerical data types, string, bool and some other data types.

- **int** - integers: ..., -3, -2, -1, 0, 1, 2, 3, ...
- **float** - floating point numbers, decimal point fractions: -3.1, 1.5, 1e-8, 3.141592e5
- **str** - character strings, text: "WhynotPython", 'python', '3.16'
- **bool** - boolean values: `True` and `False`
- **complex** - A complex number is defined using a real component + an imaginary component j

- **list** - An ordered list of items : `[ 'apple', 'orange', 1, 5.3, 'Hello']`
- **tuple** - Tuples are immutable lists, while the list data can be modified, : `(1, 3, 5)`
- **dict** - Dictionaries are made up of key: value pairs: `{"Lee": 8 , "Kay": 5}`
- **set** - An unordered collection of data type without duplicate elements: `{1, 2, 'hello }`

- **range** - range

int - Integer data type

The first of these is the integer type. We can add integers, getting another integer as the result:

```
In [ ]:
```

We can also assign an integer to a variable:

```
In [ ]:
```

We can ask the variable for its type, and see that it is `int` :

In [ ]:
```
                    # 2**8 * 2**8 = 2**16
```

In [ ]:
```
         # 2**16
```

In [ ]:
```

```

NOTE: There is no longer a limit to the value of integers.

In [ ]:
```
a = 2 ** 1000
```

In [ ]:
```
a = '123'
```

Specifying a Variable Type

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

- `int()` - constructs an integer number from an int, float, str literals
- `float()` - constructs a float number from an int, float, str literals
- `str()` - constructs a string from a wide variety of data types, including strings, integer literals and float literals

For example

- Convert `int` to `float` using `float()`

In [ ]:
```
a = '12.34'
a = None
type(None)
```

float - Floating Point Numbers

You can represent real numbers in Python as easily as integers. You may use `float()` to convert `int` to `float` data type. For example,

Notice that the type of `a` is now `float`.

This can be generalized for other numeric types: `int()`, `complex()`. It is called **type conversion**.

convert `float` to `int` using `int()`

In [ ]:
```
# float a = -1.98765 to int
a = -1.98765
```

```
b = float(None)
type(b)
```

Notice that int() type conversion always **rounds towards 0**.

Even you can convert a number in string as well.

In [ ]:
```
int('123')
```

In [ ]:
```
float('1.234')
```

Str - string data type

Another built-in Python data type is strings. Strings are sequences of letters, numbers, symbols, and spaces. In Python, strings can be almost any length and can contain spaces. Strings are assigned in Python using single quotation marks ' ' or double quotation marks " ".

- To represent a string, you may use a pair of single, double, or triple of double quations. Use a triple quote to have a multiple lines of comments.

- For example

  ```
  'Hello World'
  "Hello World"
  """Hello World"""
  'Hello "Mr. Kim"'
  ```

String Operators:

> `x in s` True if string x is a substring of string s, and false otherwise
>
> `x not in s` False if string x is a sbustring of string s, and true otherwise
>
> `s + t` Concatenation of string s and string t
>
> `s * n, n * s` Concatenation of n copies of s
>
> `s[i]` Character of string s at index i
>
> `len(s)` Length of string s

To obtain in the full list, use `help(str)` .

Example 1.

- `print()` function - prints a textual representation to the console

In [ ]:
```
a = 'Hello World'
type(a)
print(a)
```

- Multiple lines quoted by triple quartation markers shown below:
  `Hello`
  `World`

In [ ]:
```
b = None
type(b)
```

```
print(b)
```

Example 2. Make an observation of three different outputs

In [ ]:
```
print("Hello World")
```

In [ ]:
```
print("Hello", 'World')
```

In [ ]:
```
print("Hello" + 'World')
```

Example 3:

How do you print the follwoing sentence as shown exactly?

```
Hello "Mr. Kim"
```

In [ ]:
```
print(None)
```

Example 4:

How do you print the follwoing sentence as shown exactly?

```
----------------------------------
| Hello World, God loves you all. |
----------------------------------
```

In [ ]:
```
print(None)
```

Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.
- However, Python does not have a character data type, a single character is simply a string with a length of 1.
- The indivisual characters of a string can be accessed using the **indexing operators []**, a pair of square brackets.
- Negative indexes may be used to access the characters from back (right side) of the string.

Example 1:

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

In [ ]:
```
a = "Hello, World!"
print(a[1])
```

Example 2: Slicing

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

In [ ]:
```
b = "Hello, World!"
print(None)
```

Example 3: Negative Indexing

- Use negative indexes to start the slice from the end of the string
- Get the characters from position 5 to position 1 (not included), starting the count from the end of the string:

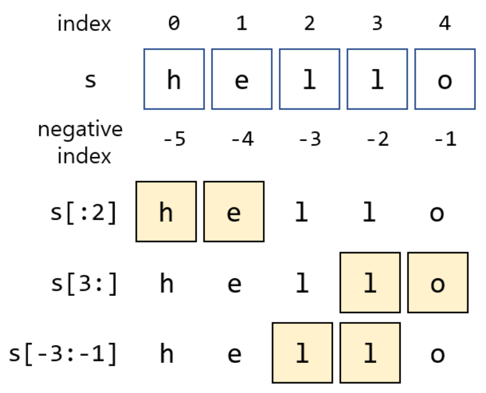


Figure 2. Negative Indexing

In [ ]:
```
b = "Hello, World!"
print(None)
```

Arithmetic Operations

Let's use an example of a more involved numerical expression to get some familiarity with Python's operation syntax and order of operations.

Evaluate `1 - 4 * 2**3 - 60 // 7 + 89 % 98`

Looking at this, most of the operations should be straightforward, but `**` in Python indicates exponentiation, and `%` is the modulo (or remainder) operation.

Python's precedence order for operations is fairly standard (from highest to lowest):

> `( )` (parenthesis grouping)
> `**` (exponentiation)
> `*` (multiplication)
> `/` (true division)
> `//` (integer division) - also called **floor division** since applying the floor function after division.
> `%` (modulus) - remainder of the division of the two operands, rather than the quotient
> `+, -` (addition, subtraction)

Example 1: `23 // 4` evaluates to `5`.

Example 2: `23 % 4` evaluates to `3` because there's a remainder of 3.

Example 3: `4*2**3` evaluates the exponentiation part first and multiplication.

In [ ]:
```
None
```

So we now have `1 - 32 - 60 // 7 + 89 % 98`.

The next operations are multiplication, integer division // and modulo. Be caucious when there is a negative number in modulo operation.

In [ ]:
```
None
```

and the expression simplifies to `1 - 32 + 8 + 89`.

In [ ]:
```
print(1 - 4*2**3 - 60 // 7 + 89 % 98)

print(1 - 32 - 8 + 89)
```

Evaluating expressions:

Example 1:

Print each digit in a three-digit number.

In [ ]:
```
digits = 248
digit1 = None
digit2 = None
digit3 = None

print(digit1)
print(digit2)
print(digit3)
```

Example 2:

Evaluate the expression.

In [ ]:
```
((2 * 3 + 4) / 5) ** 2 % 3
```

Example 3:

Write a code that accepts an age from user and display the result as shown below. You may need to use `int()` and `input()`.

```
What is your age?
7
Your age becomes 8 tomorrow.
```

In [ ]:
```
age = None
print('Your age becomes', None, 'tomorrow. Happy Birthday!')
```

Simple Math Functions

There are a number of simple math functions that are available in Python, and we'll show a few here to give you an idea.

```
abs()
round()
max()
min()
```

Example 1.

You can get the absolute value with `abs()` :

In [ ]:
```
abs(None)
```

Example 2.

You can round a floating point number to the nearest whole number using `round()` :

In [ ]:
```
import numpy as np
np.sqrt(3)
```

Example 3.

And you can use `max()` and `min()` functions to compute minimum and maximum values from a collection of values:

In [ ]:
```
max(0, min(10, -1, 4, 3))
```

This example is a little more involved, what's happening is that we have two parts.

- The first is the minimum of `10` , `-1` , `4` and `3` .
- The second is the maximum of 0 and the result of the first part which is 0, and that's the result.

Example 4.

Find the max in the list.

In [ ]:
```
colors = ['red', 'orange', 'yellow', 'green', 'navy blue', 'indigo', 'violet']
primes = [11, 13, 2, 3, 5, 7]
```

Example 5.

Find the max string length among colors.

- The built-in `max()` function returns the maximum element in the list. The elements could be int, float, or string.
- To get the longest word, use `max() with key=len` argument. It is passing a comparison function as an argument.
- help(max)

In [ ]:
```
max(colors, None)
```

Using `math` module

Besides built-in math functions, you may import `math` module first to use. Listed are some functions and constants in the `math` module. After importing the module, you can obtain full list using `help(math)` .

> `sqrt(x)` $\sqrt{x}$
> `ceil(x)` $\lceil x \rceil$ (i.e., the smallest integer $\geq x$)
> `floor(x)` $\lfloor x \rfloor$ (i.e., the largest integer $\leq x$)
> `cos(x)` $cos(x)$
> `sin(x)` $sin(x)$
> `log(x, base)` $log_{base}(x)$
> `pi` $3.141592653589793``e`$2.718281828459045`

bool - Boolean Data Type

- In programming you often need to know if an expression is True or False.
- You can evaluate any expression in Python, and get one of two answers, True or False.
- When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Example 1:

In [ ]:
```
print(10 > 9)
print(10 == 9)
print(10 < 9)
```

When you run a condition in an if statement, Python returns True or False:

Example 2:

In [ ]:
```
a = 200
b = 33

if b > a:
    print("b is greater than a")
else:
    print("b is not greater than a")
```

Boolean operators:

```
or
and
not
```

Comparison and Boolean Operators:

Comparison Operators:

> `>` Greater than - True if left operand is greater than the right `x > y`
> `<` Less than - True if left operand is less than the right `x < y`
> `==` Equal to - True if both operands are equal `x == y`
> `!=` Not equal to - True if operands are not equal `x != y`
> `>=` Greater than or equal to - True if left operand is greater than or equal to the right `x >= y`
> `<=` Less than or equal to - True if left operand is less than or equal to the right

Boolean Operators:

> `not x` Boolean NOT
> `and` Boolean AND
> `or` Boolean OR

These comparison and boolean operators are quite standard. So for example:

In [ ]:
```python
# Evalueate the result of a boolean expression
a = 3 > 4
a
```

and see that the type of `q` is now `bool` :

In [ ]:
```python
# Evaluate p and q and print whetehr or not it is true.
q = 5 > 10
p = 5 < 10
if (None):
    print('q is true')
```

Boolean(Logical) Expressions

A boolean expression (or logical expression) evaluates to one of two states true or false. Comparisons can be combined using the logical operators `and` , `or` and `not` . So for example:

Example 1:

Write a boolean expression that returns true if `age` is between `20` and `29` , inclusively, false otherwise.

In [ ]:
```python
None
```

Example 2: Using chained comparison

- One really **slick feature** of Python's comparisons are chained comparisons:

In [ ]:
```
None
```

Example 3:

What is the result of the following expression?

In [ ]:
```
5 < 10 and 10 < 20
```

In [ ]:
```
5 < 10 or 10 < 20
```

In [ ]:
```
5 < 10 or 10 > 20
```

Short-circuit evaluation

Python uses "short-circuit" evaluation of logical expressions if the value of subsequent terms can't affect the result.

Example 4:

In [ ]:
```
1 < 0 and max(0, 1, 2) > 1
```

`False` since the first operand is `False` regrardless of the following part of expression.

There is no way that the full expression could ever evalute to `True` and so Python **skips the evaluation of the second operand**, and returns `False` straight away. This is called **short-circuit evaluation**.

Example 5:

What do you expect from the following two logical expressions, respectively?

1. `1 > 4 and 5 < xx`
2. `1 > 4 or 5 < xx`

Possible choices:

1. Name error
2. Type error
3. True
4. False

In [ ]:
```
1 > 4 and 5 < xx
```

In [ ]:
```
1 > 4 or 5 < xx
```

Example 6:

What is the evaluation of the following expreesion?

In [ ]:
```
a = 50
a < 10 or a > 100
```

Python's `or` operator works in a similar way. The short-circuit evaluation works here since the first operand is `True` :

Example 7:

What is the evaluation of the following expreesion?

In [ ]:
```
b = 0
b < 10 or b > yy    # second term won't be evaluated
```

And finally, the `not` operation inverts the value of the argument:

In [ ]:
```
not 10 <= a <= 90
```

Example 8:

Write a logical expression which evaluates an age to be `True` if the age is a teenage. The teenage means between 13 and 19 inclusively.

In [ ]:
```
age = 15
# write a logical expression below to determine whether or not age is teen.

13 <= age <= 19
```

Range is a special object in Python

- Used to generate integer numbers within a range of values
- Can iterate through the range

In [ ]:
```
for x in range(0, 5):
    print(x)
```

Exercises

특별한 지시가 없는 한, 다음 문제들 중에서 여러 방법으로 코딩할 수 있는 문제는 단 하나의 방법으로만 코딩해도 됩니다.

Given an integer number, print its last digit.

Print the last digit of `a` which is an integer.

Sample Runs:

```
Enter an integer number:  369
<class 'str'>
9

Enter an integer number:  6
<class 'str'>
6

Enter an integer number:  9765
<class 'str'>
5
```

Method 1: Using any arithmetic operators.

- Often we use both modulus `%` operator and floor division `//` .

In [ ]:
```
# Read an integer:
a = input("Enter an integer number: ")
print(type(a))
# using mod % operator
```

Method 2: Using `len()` and indexing

In [ ]:
```
# Read an integer as a string
s = input("Enter an integer number: ")
print(type(s))
# Using len() and indexing
```

Method 3: Using indexing

In [ ]:
```
# Read an integer as a string
s = input("Enter an integer number: ")
print(type(s))
# using indexing
```

Given a three-digit number. Find the sum of its digits.

Print the sum of three-digit number which is an integer.

Sample Runs:

```
Enter a three-digit number: 316
10

Enter a three-digit number: 256
13

Enter a three-digit number: 567
18
```

Method 1: Using % (mod operator) and // (integer division)

In [ ]:
```
a = int(input("Enter a three-digit number:"))
x = None                 # get 100's
y = None                 # get 10's
z = None                 # get 1's
print (x + y + z)
```

In [ ]:
```
a = int(input("Enter a three-digit number:"))
x = None                 # get 100's
y = None                 # get 10's
z = None                 # get 1's
print (x + y + z)
```

Clock displays military time

The integer N given is the number of minutes that is passed since midnight. Display the hour and minute in military time. The program should print two numbers: the number of hours (between 0 and 23) and the number of minutes (between 0 and 59).

For example, if N = 150, then 150 minutes have passed since midnight - i.e. now is 02:30. So the program should print 02:30. Diplay 16:40 for 1000, 09:20 for 2000.

Sample Run:

```
15
00:15

180
03:00

1000
16:40

2000
09:20
```

In [ ]:
```
# Read an integer:
a = int(input())
print(a)
# print 03:00 for 180, 07:24 for 444, 18:31 for 1111, 23:59 for 1439

None
```

Equal numbers

Given three integers, determine how many of them are equal to each other. The program must print one of these numbers: 3 (if all are the same), 2 (if two of them are equal to each other and the third is different) or 0 (if all numbers are different).

Sample Run:

```
x:1
y:2
z:3
0

x:3
y:3
z:5
2

x:9
y:9
z:9
3
```

In [2]:

```python
# Read an integer:
x = int(input("x:"))
y = int(input("y:"))
z = int(input("z:"))
None
```

참고자료

- Python Data Types

학습 정리

1. 파이썬 데이터 타입 살펴보기
2. 데이터 타입별 사용방법 익히기

The LORD bless you and keep you. Numbers 6:24