Operations using Data Types and Operator

Pandas Group – Data Science Track B

My Edu Solve

Ajie Prasetya | Dewa Ayu Rai I.M. | I K. Juni Saputra | Firdausi Nurus Sa'idah | M. Ramadhoni | Vivi Indah Fitriani

Data Type Conversion

1. Converting Integers to Float

To convert the integer to float, use the float() function in Python. Similarly, if you want to convert a float to an integer, you can use the int() function.

```
[ ]  x = 22
     print(type(x))

     <class 'int'>

[ ]  x = 22
     float(x)

     22.0

[ ]  print(type(float(3)))

     <class 'float'>
```

2. Converting Float to Intenger

Converting float to int will be floor/truncating or omit the value after the comma

```
✓ 0s [25]  z = 165.0
           type(z)

           float

✓ 0s [26]  int(z)

           165

✓ 0s       print(type(int(z)))

           <class 'int'>
```

3. Converting Numbers to Strings

We can convert numbers to strings through using the str() method. We'll pass either a number or a variable into the parentheses of the method and then that numeric value will be converted into a string value.

```
[28] #Converting Integers to Strings

     price = 15000
     type(price)

     int

[29] str(price)

     '15000'

[30] print(type(str(price)))

     <class 'str'>
```

4. Converting Strings to Numbers

We can converting strings to numbers. For example we want to convert strings to float data type

```
[31] price_mouse = "95000.63"
     price_flashdisk = "79000.49"
     type(price_mouse)
     type(price_flashdisk)

     str

[32] # Explicit type conversion to float
     total = float(price_mouse) + float(price_flashdisk)
     print("The total is: " + "Rp." + str(total))

     The total is: Rp.174001.12
```

5. Converting List to Tuples

Just like integers and floats, we can also convert lists to tuples and tuples to lists. We can use the tuple() function to return a tuple version of the value passed to it.

```
[ ] flowers = ["roses", "sunflowers", "aster", "azalea", "jasmine", "peonies"]
    print(type(flowers))

    <class 'list'>

[ ] tuple(flowers)

    ('roses', 'sunflowers', 'aster', 'azalea', 'jasmine', 'peonies')

[ ] print(type(tuple(flowers)))
```

6. Converting Tuples to List

We can use the list() method to convert the following tuple to a list. Because the syntax for creating a list uses parentheses, make sure to include the parentheses of the list() method, and in this case the print() method as well:

```
[ ] identity = ("Salsabila Grace", "21", "Jakarta", "Universitas Pendidikan Indonesia")
    type(identity)

    tuple

[ ] list(identity)

    ['Salsabila Grace', '21', 'Jakarta', 'Universitas Pendidikan Indonesia']

[ ] print(type(list(identity)))

    <class 'list'>
```

7. Converting to Set

We can converting data type list to set. For example:

```
[ ]  hp_brand = ["Apple","Xiaomi","Samsung","Oppo"]
     type(hp_brand)

     list

[ ]  set(hp_brand)

     {'Apple', 'Oppo', 'Samsung', 'Xiaomi'}

[ ]  print(type(set(hp_brand)))

     <class 'set'>
```

8. Converting to Dictionary

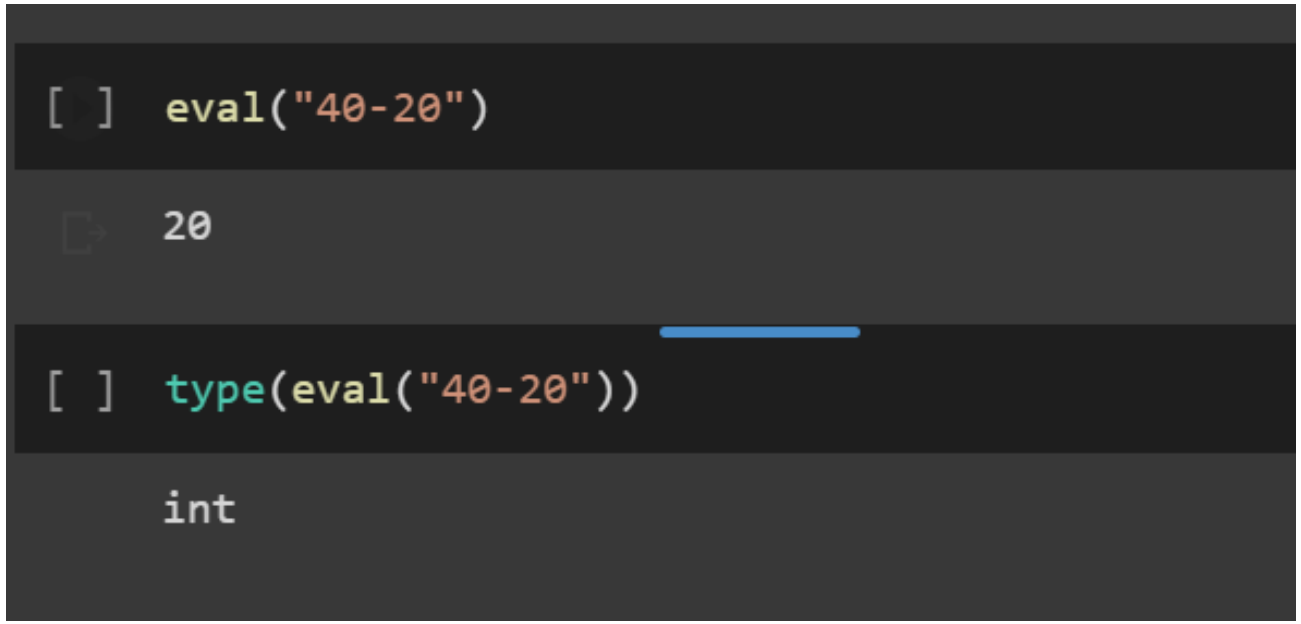
If we want to converting data type to a dictionary, the data must meet the key-value requirements. Here are two examples of conversions:

- A list of several lists containing value pairs becomes a dictionary
- As well as converting a List of several tuples containing value pairs into a dictionary

```
[ ]  #Converting a list that containing value pairs to a dictionary

     [[7,8],[2,3]]
     dict([[7,8],[2,3]])

     {2: 3, 7: 8}

[ ]  #Converting list of several tuples to a dictionary

     ((4,5),(6,7))
     dict(((4,5),(6,7)))
```

Evaluation Function

If the input is a string containing a mathematical expression, then conversion to int() or float() will return an error. You can use the eval() function which also solves mathematical expressions



Input and Output Operation

1. Variable

Variable is a place (in computer memory) to store values of a certain data type. To assign a value to a variable, we use the "=" operator, between the variable name and the value we want to store.

```
[ ]  x = 50

     #It means that we will store the value 50 (int type) into the variable x
```

2. Print

The print() function prints the given object to the standard output device (screen) or to the text stream file.

```
program = "Data Science Track"
print(program)
```

```
Data Science Track
```

Entering a variable value in a string

Python has several ways to include variable values in strings there are:

1. Directly concatenate variables in the print() statement

```
[ ]  a = 9
     print('The value of variable a is', a)

     The value of variable a is 9
```

2. Formatting

Sometimes we would like to format our output to make it look attractive. This can be done by using the .format() method. This method is visible to any string object.

```
[ ]  print("Hello! My name is {}".format("Seo Dal Mi"))

     Hello! My name is Seo Dal Mi
```

Entering a variable value in a string

3. Using "%" operator added with "Argument specifiers"

- We can also format strings like the old sprintf() style used in C programming language. We use the % operator to accomplish this.
- Some examples of commonly used argument specifiers:
    1. %s - String
    2. %d - Integers
    3. %f - Float(Decimal Number)

```
[9]  nama = "Justin Bieber"
     umur = 28
     tinggi = 178

     print ("Hello %s, saat ini umurmu %d tahun dan tinggi badanmu %f cm" %(nama, umur, tinggi))
```

```
Hello Justin Bieber, saat ini umurmu 28 tahun dan tinggi badanmu 178.000000 cm
```

Function Input()

To allow the user to provide input to your code program, use the input() function where the argument in brackets() is the text you want to display (the prompt) and the variable before the equal sign(=) is the resultant holder of the user's input:

```
[ ]  num1 = int(input("Enter first number : "))
     num2 = int(input("Enter second number : "))
     result = num1* num2
     print("The result of the multiplication is", result)
```

```
Enter first number : 6
Enter second number : 7
The result of the multiplication is 42
```

List, Set, and String Operations

1. List Operation

List responds to operators + and * like strings. That means the merger and repetition here also applies, unless the result is a new list, not a String.

```
[ ]  Item = ['table', 'chair', 'cupboard', 'window']
     number = [1,2,3,4,5]

[ ]  print(len(Item)) #indicates many objects on the list.
     print(Item+number) #Adding objects on item and number
     print(number*2) #multiplication list in number twice
     print(3 in number) #Check if 3 is in the list number

     4
     ['table', 'chair', 'cupboard', 'window', 1, 2, 3, 4, 5]
     [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]
     True
```

2. Set Operations

In set, the operations performed are useful for the data processing process, that is union function (combined), intersection function (slice), difference function (difference), symmetric difference function (complement)

```
[ ]  Age1 = {22, 13, 17, 14}
     Age2 = {20, 24, 17, 22}

[ ]  print(Age1.union(Age2)) #combine data on age1 and age2
     print(Age1.intersection(Age2)) #Take data who belong to age1 and age2
     print(Age1.difference(Age2)) #retrieve data on age1 that is not data in age2
     print(Age1.symmetric_difference(Age2)) #Displays data that is only in one variabel

     {13, 14, 17, 20, 22, 24}
     {17, 22}
     {13, 14}
     {13, 14, 20, 24}
```

3. Strings Operations

Strings can be expressed using single quotation marks ('), double quotation marks (""), three pairs of single quotation marks ('''''), or three pairs of double quotation marks (""""""). String operations using +, and *operators, and using the len, max, min functions can be executed as in the list or tuple.

```
[ ]  info1 = "This week is the sixth week of Independent study activities. "
     info2 = "In this week there is also a lot of new science that my colleagues and I have learned."

[ ]  print(info1 + info2) #adding the sentences in info1 and info2
     print(len(info1)) #count the length of a character in info1
     print(info1*2)  #multiply the sentence in info1 by 2 times
     print(max(info2)) #search for the largest alphabetical order of sentences in info2

     This week is the sixth week of Independent study activities. In this week there is also a lot of new science that my colleagues and I have learned.
     61
     This week is the sixth week of Independent study activities. This week is the sixth week of Independent study activities.
     y
```

Dynamic Typing

Python applies one of the languages, namely dynamic typing, which is a programming language that only knows the type of variables when the program is running and assignments are made.

```
[ ]  a = "My name is Lupin"
     type(a) # Show varible type data

     str
```

```
[ ]  len(a) # Indicates the length of the string and space, but doesn't work to numeric variables (ex. integers)

     16
```

Operator, Operand, and Expressions

Operator is an instruction given to get the results of a process carried out.

Operand is the origin value used in an operation process. Expression is a syntactic entity that gen erates a value, where usually the expression can consist of operator and operand.

For example, in operation 1 + 2. The numbers 1 an d 2 are operands, the + sign is referred to operator, while 1 + 2 is referred to as the expression.

There are various types of operators used in the Pyhton, as follows.

1. Arithmetic Operator (Operator Aritmatika),
2. Comparison Operator (Operator Perbandingan),
3. Assignment Operator (Operator Penugasan),
4. Logical Operator (Operator Logika),
5. Bitwise Operator (Operator Bitwise),
6. Membership Operator (Operator Membership),
7. Identity Operator (Operator Identitas).

1. Arithmetic Operator

Arithmetic Operator is an operator used to perform mathematical operations such as sum, subtraction, division, multiplication, and others.

```
1 a = 7
2 b = 3
```

```
[ ]  1 print(a+b) #(+), indicates sum
     2 print(a-b) #(-), indicates subtraction
     3 print(a/b) #(/), indicates division
     4 print(a*b) #(*), indicates multiplication
     5 print(a**b) #(**), indicates power
     6 print(a//b) #(//), indicates division (rounded down)
     7 print(a%b) #(%), indicates enrichment

    10
    4
    2.3333333333333335
    21
    343
    2
    1
```

2. Comparasion Operator

Comparison operators are used to compare 2 values. The result is true or false

```
[ ]  1 x = 4
     2 y = 2
     3 z = 7
```

```
[ ]  1 print(x == y)  #(==), indicates equal to
     2 print(x != z)  #(!=), indicates not equal to
     3 print(z > x)   #(>), indicates greater than
     4 print(y >= x)  #(>=), indicates greater than or equal to
     5 print(y < z)   #(<), indicates less than
     6 print(z <= y)  #(<=), indicates less than or equal to

    False
    True
    True
    False
    True
    False
```

3. Assignment Operator

Assignment operators are used to assign values to variables. In Python, the assignment operator uses the same sign as (=). Assignment operators also have a writing variation referred to as a compound assignment operator.

```
[ ]  1 y=12
     2 y-=7  #Subtract y with 7
     3 print(y)

     5
```

```
[ ]  1 z=30
     2 z/=3  #Div z with 3
     3 print(z)

     10.0
```

4. Logical Operator

Logic operator is an operator used in logic operations, such as and, or, or note logic.

```
[ ]  1 w = 4
     2 x = 4
     3 y = 7
     4 z =10
```

```
[ ]  1 print(w > 1 and y > 1)  #if both are true, the result will be true.
     2 print(w < 1 and y < 1)  #if one of expressions is false then the result will be false
     3 print(y < z or w > x)   #if one is true, the result will be true.
     4 print(y > z or w < x)   #if both are false, then the result will be false.
     5 print(not(w > 1 and y > 1))  #if the result is true, then it will return the false, and vice versa

     True
     False
     True
     False
     False
```

5. Identity Operator

Identity operators are used to compare objects, whether they are the same object or not.

```
[ ]  1 a = 8
     2 b = 8
     3 c = 4

[ ]  1 a is b #Returns True if both variables are the same object

     True

[ ]  1 a is c #Returns True if both variables are the same object

     False

[ ]  1 a is not b #Returns True if both variables are not the same object

     False
```

6. Membership Operator

Membership operators are used to test whether a sequence is presented in an object.

```
1 Animals = ['Chicken', 'Cat', 'Dear', 'Lion', 'Crocodile']

[ ]  1 print('Chicken' in Animals) #returns true when on the list, and false if not on the list

     True

[ ]  1 print('Dog' in Animals) #returns true when on the list, and false if not on the list

     False

[ ]  1 print('Crocodile' not in Animals) #returns true when the data is not on the list, but return false if the data on the list

     False
```

7. Bitwise Operator

Bitwise is a special operator for handling the logical operation of binary numbers in the form of bits. Bi twise operators are not used very often, unless you are creating a program that must process the bits of the computer. In addition, these operators are quite complicated and must have an understandin g of binary number systems.

```
1 print(8 & 4) #if the two bits are equally 1, then the result is also 1, in addition to the condition, the final value is 0. 8 = 1000 and 4 = 0100, so the resu
2 print(8 | 4) #the result will be 0 if both bits are 0, in addition the bit value will be set to 1. 8 = 1000 and 4 = 0100, so the result is 12 = 1100
3 print(8 ^ 4) #if the two bits are different, the result is 1 but if it is both 0 or equally 1, the result is 0.
```

```
0
12
12
```

Transform Number, Characters, and String

Case Manipulators:

1. .upper()
2. .lower()

String Methods:

1. .rstrip()
2. .lstrip()
3. .strip()

Testing for Substring-hood:

1. .starswith()
2. .endswith()

1. .upper()

The upper() methods returns the uppercased str ing from the given string. It converts all lowercas e characters to uppercase. If no lowercase char acters exist, it returns the original string.

```
[ ]  1 name = "rani andini"
     2 name.upper() # Converting string into it's upper case

'RANI ANDINI'
```

2. .lower()

The lower() methods returns the lowercased stri ng from the given string. It converts all uppercas e characters to lowercase. If no uppercase chara cters exist, it returns the original string.

```
[ ]  1 now = "It's Raining"
     2 now.lower() # Converting string into it's lower case

'it's raining'
```

3. .rstrip()

This method is used to delete whitespace or cha racters mentioned to the right of a string o ending of a string.

```
[ ]  1 name = "   My name's Tom Holland    "
     2 name.rstrip() # Delete all whitespace in the right of a string

'   My name's Tom Holland'
```

4. .lstring()

This method is used to delete whitespace or charact ers mentioned to the left of a string or beggining of a string.

```
[ ]  1 home = "   My house in Los Angeles    "
     2 home.lstrip() # Delete all whitespace in the left of a string

'My house in Los Angeles    '
```

5. .strip()

This method is used to delete whitespace at the be ginning and end of a string, strip() can also specify which characters or parts you want to deleted.

```
[ ]   1 pathway = "----Data Science----"
      2 pathway.strip("-") # Delete all '-'

     'Data Science'
```

6. .startswith()

The purpose of this function is to return true if the function begins with mentioned string (prefix) el se return false.

```
[ ]   1 food = "Ketoprak is Indonesian food"
      2 food.startswith("Ketoprak") , food.startswith("Indonesian")

     (True, False)
```

7. .endswith()

The purpose of this function is to return true if the function ends with mentioned string (suffix) else return false.

```
[ ]   1 food = "Ketoprak is Indonesian food"
      2 food.endswith("food") , food.endswith("Ketoprak")

     (True, False)
```

Conditional Expression

A conditional expression is a tested conditional statement that can contain mathematical operators, functions, and logic. Each conditional expression ends with a colon ":". Each conditional expression has a command code in writing indents.

If Function

```
[ ]  vaccine = "Booster"

     if vaccine:
       print("The 3rd vaccine is {}".format(vaccine))

     The 3rd vaccine is Booster
```

If Else Function

```
vaccine=str(input("Vaccine Type: "))

if vaccine == "Booster":
  print("Goo 'Mudik'")
else:
  print("No 'Mudik'")

Vaccine Type: Booster
Goo 'Mudik'
```

Elif Function

```
vaccine=str(input("Name: "))
vaccine=str(input("Vaccine Dose: "))

if vaccine == "Third Dose":
  print("full face-to-face lecture")
elif vaccine == "Second Dose":
  print("face-to-face lecture 50%")
else:
  print("Online Lecture'")

Name: Budi
Vaccine Dose: Third Dose
full face-to-face lecture
```

Replacing String Elements

replace() is an inbuilt function in the Python programming language that returns a copy of the string where all occurrences of a substring are replaced with another substring.

```
[ ]  a = "sekarang kita sekolah"

[ ]  print(a.replace("sekolah","tidak"))

     sekarang kita tidak
```

the third parameter in replace can be filled with the number of substrings you want to replace

```
[ ]  a = "sekarang kita sekolah dan saya membawa bekal"

[ ]  print(a.replace("sekolah","bimbel", 1))

     sekarang kita bimbel dan saya membawa bekal
```

String Checking

The in operator is the easiest and pythonic way to check if a python string contains a substring. The in and not in are membership operators, they take in two arguments and evaluate if one is a member of the other. They return a boolean value.

islower()

```
[ ]  a = "metodologi penelitian"

[ ]  a.islower()
     True

[ ]  print("metodologi".lower().upper())
     METODOLOGI

[ ]  print("metodologi".upper().lower())
     metodologi
```

issuper()

```
[ ]  a = "Metodologi Penelitian"

     a.isupper()
     False
```

isalnum()

```
[ ]  print("Metodologi".isalnum())
     True
```

isdecimal()

```
[ ]  print("Metodologi Penelitian".isdecimal())
     False
```

isalpha()

```
     print("metodologi".isalpha())
     True

[ ]  print("duze".isalpha())
     True
```

String Formatting

string.rjust()

```
[ ]  data = "Data Science".rjust(20)

[ ]  print("Before: \nData Science")
     print("\nAfter: \n", data)

     Before:
     Data Science

     After:
              Data Science

[ ]  data

     '        Data Science'

     len("Data Science")

     12

[ ]  len(data)

     20
```

string.ljust()

```
[ ]  text = "Data Science"
     leftadjust_text = text.ljust(20)

[ ]  leftadjust_text

     'Data Science        '

     print("Before: \n", text)
     print("\nAfter: \n", leftadjust_text)

     Sebelumnya:
      Data Science

     Sesudah:
      Data Science

[ ]  print("length of text before: ", len(text))
     print("length of text after adjust: ", len(leftadjust_text))

     length of text before:  12
     length of text after adjust:  20
```

string.center()

```
[ ]  course = "Data Science"

[ ]  center_course = course.center(20)
     center_course

     '    Data Science    '

     print("length text before adjustment: ", len(course))
     print("length text afrer adjustment: ", len(center_course))

     length text before adjustment:  12
     length text afrer adjustment:  20

[ ]  center_course = course.center(20,",")
     print(center_course)

     ,,,,Data Science,,,,
```

String Literals

```
[ ]  print("aren't ")

     aren't

[ ]  print('aren't)

       File "<ipython-input-20-6762bc4af60b>", line 1
         print('aren't)
                     ^
     SyntaxError: invalid syntax

     SEARCH STACK OVERFLOW

     print('aren\'t')

     aren't
```

Raw String

```
[ ]  print(r'aren\'t')

     aren\'t

[ ]  text_tab = "Data \tScience"

     print("Value showed :")
     print(text_tab)
     print("\nRaw strings :")
     print(r"Data \tScience")

     Value showed :
     Data    Science

     Raw strings :
     Data \tScience
```

Thank You