Programming with Python

Understanding Data Types in Python.

Kuo, Yao-Jen yaojenkuo@datainpoint.com (mailto:yaojenkuo@datainpoint.com) from DATAINPOINT (https://www.datainpoint.com/)

TL; DR

In this lecture, we will talk about operators, variables, and types.

Arithmetic Operators in Python

Don’t know what to do with Python?

Source: https://giphy.com/ (https://giphy.com/)

Use it as a calculator

- +, –, *, / are quite straight-forward
- ** for exponentiation
- % for remainder
- / / for floor-divide

When an expression contains more than one operator, the order of evaluation depends on the operator precedence

1. Parentheses have the highest precedence.
2. Exponentiation has the next highest precedence.
3. Multiplication and Division have higher precedence than Addition and Subtraction.
4. Operators with the same precedence are evaluated from left to right.

For example, calculating BMI given height in centimeters and weight in kilograms.

$$BMI = \frac{weight_{kg}}{height_{m}^{2}}$$

In [1]:
```
70/175/100**2
```

Out[1]:
```
4e-05
```

In [2]:
```
70/(175/100)**2
```

Out[2]:
```
22.857142857142858
```

For example, converting a degree of Farenheit scale to Celsius scale.

$$Celsius^{\circ}C = (Fahrenheit^{\circ}F - 32) \times \frac{5}{9}$$

In [3]:
```
212 - 32*5/9
```

Out[3]:
```
194.22222222222223
```

In [4]:
```
(212 - 32)*5/9
```

Out[4]:
```
100.0
```

Variables

One of the most powerful features of a programming language is the ability to manipulate variables

A variable is a name that refers to a value.

Choose names for our variables: DON'Ts

- Do not use built-in functions
- Cannot use keywords (https://docs.python.org/3/reference/lexical_analysis.html#keywords)
- Cannot start with numbers

If you accidentally replaced built-in function with variable, use `del` to release it

In [5]:
```
print = 5566
print("Hello, world!")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-55e8932a6551> in <module>
      1 print = 5566
----> 2 print("Hello, world!")

TypeError: 'int' object is not callable
```

In [6]:
```
del print
print("Hello, world!")
```

```
Hello, world!
```

Choose names for our variables: DOs

- Use a lowercase single letter, word, or words
- Separate words with underscores to improve readability(so-called snake case)
- Be meaningful

Using # to write comments in our program

Comments can appear on a line by itself, or at the end of a line.

In [7]:
```
degree_f = 50
# turn fahrenheit into celsius
degree_c = (degree_f - 32) * 5/9
degree_c = (degree_f - 32) * 5/9 # turn fahrenheit into celsius
print(degree_c)
```

```
10.0
```

Everything from # to the end of the line is ignored when executed

We can use pythontutor.com (http://www.pythontutor.com/visualize.html#mode=edit) to explore the execution of our code.

Common Data Types

Values belong to different types, we commonly use

- `int` and `float` for computing
- `str` for symbolic
- `bool` for conditionals
- `None` for missing values

Use `type` function to check the type of a certain value/variable

In [8]:
```
print(type(5566))
print(type(42.195))
print(type("Hello, world!"))
print(type(True))
print(type(False))
print(type(None))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
<class 'bool'>
<class 'NoneType'>
```

How to form a `str`?

Use paired `'`, `"`, or `"""` to embrace letters strung together.

In [9]:
```
str_with_single_quotes = 'Hello, world!'
str_with_double_quotes = "Hello, world!"
str_with_triple_double_quotes = """Hello, world!"""
print(type(str_with_single_quotes))
print(type(str_with_double_quotes))
print(type(str_with_triple_double_quotes))
```

```
<class 'str'>
<class 'str'>
<class 'str'>
```

If we have single/double quotes in `str` values

In [10]:
```
mcd = 'I'm lovin' it!'
```

```
  File "<ipython-input-10-85a683c7c2bf>", line 1
    mcd = 'I'm lovin' it!'
            ^
SyntaxError: invalid syntax
```

Use \ to escape or paired " or paired """

In [11]:
```
mcd = 'I\'m lovin\' it!'
mcd = "I'm lovin' it!"
mcd = """I'm lovin' it!"""
```

We've seen arithmetic operators for numeric values

How about those for `str` and `bool` ?

`str` type takes + and *

- + for concatenation
- * for repetition

In [12]:
```
mcd = "I'm lovin' it!"
print(mcd)
print(mcd + mcd)
print(mcd * 3)
```

```
I'm lovin' it!
I'm lovin' it!I'm lovin' it!
I'm lovin' it!I'm lovin' it!I'm lovin' it!
```

Format our `str` printings

- The `sprintf` way
- The `.format()` way
- The `f-string` way

The `sprintf` way: uses `%` for string print with format

In [13]:
```
my_name = "John Doe"
print("Hello, %s!" % (my_name))
```

```
Hello, John Doe!
```

The `.format()` way: uses `{}` for string print with format

In [14]:
```
my_name = "John Doe"
print("Hello, {}!".format(my_name))
```

```
Hello, John Doe!
```

The `f-string` way: uses `{}` for string print with format

In [15]:
```
my_name = "John Doe"
print(f"Hello, {my_name}!")
```

```
Hello, John Doe!
```

I myself, am more of a `.format()` way guy

It can take both index and key-value besides order.

In [16]:
```
print("{} {} is my favorite Friends character.".format("Phoebe", "Buffay")) # form
at with order
print("{1} {0} is my favorite Friends character.".format("Buffay", "Phoebe")) # fo
rmat with index
# format with key-value
print("{first_name} {last_name} is my favorite Friends character.".format(last_nam
e="Buffay", first_name="Phoebe"))
```

```
Phoebe Buffay is my favorite Friends character.
Phoebe Buffay is my favorite Friends character.
Phoebe Buffay is my favorite Friends character.
```

How to form a `bool`?

- Use keywords `True` and `False` directly
- Use relational operators
- Use logical operators

Use keywords `True` and `False` directly

In [17]:
```
print(True)
print(type(True))
print(False)
print(type(False))
```

```
True
<class 'bool'>
False
<class 'bool'>
```

Use relational operators

We have `==`, `!=`, `>`, `<`, `>=`, `<=`, `in`, `not in` as common relational operators to compare values.

In [18]:
```
print(5566 == 5566.0)
print(5566 != 5566.0)
print('56' in '5566')
```

```
True
False
True
```

Use logical operators

- We have `and`, `or`, `not` as common logical operators to manipulate `bool` type values
- Getting a `True` only if both sides of `and` are `True`
- Getting a `False` only if both sides of `or` are `False`

In [19]:

```
print(True and True)  # get True only when both sides are True
print(True and False)
print(False and False)
print(True or True)
print(True or False)
print(False or False) # get a False only when both sides are False
# use of not is quite straight-forward
print(not True)
print(not False)
```

```
True
False
False
True
True
False
False
True
```

Besides `type` function, the `isinstance` function can help us check if a variable stores a certain type

In [20]:
```
help(isinstance)
```

```
Help on built-in function isinstance in module builtins:

isinstance(obj, class_or_tuple, /)
    Return whether an object is an instance of a class or of a subclass thereo
f.

    A tuple, as in ``isinstance(x, (A, B, ...))``, may be given as the target
to
    check against. This is equivalent to ``isinstance(x, A) or isinstance(x,
B)
    or ...`` etc.
```

In [21]:

```
var = 5566
print(isinstance(var, int))
print(isinstance(var, float))
print(isinstance(var, str))
print(isinstance(var, bool))
```

```
True
False
False
False
```

`bool` is quite useful in programs in conditional statements, iteration, and filtering data

Python has a special type, the `NoneType`, with a single value, None

- This is used to represent null values or nothingness
- It is not the same as `False`, or an empty string `''` or 0
- It can be used when we need to create a variable but don't have an initial value for it

In [22]:

```
a_none_type = None
print(type(a_none_type))
print(a_none_type == False)
print(a_none_type == '')
print(a_none_type == 0)
print(a_none_type == None)
```

```
<class 'NoneType'>
False
False
False
True
```

Data types can be dynamically converted using functions

- `int()` for converting to `int`
- `float()` for converting to `float`
- `str()` for converting to `str`
- `bool()` for converting to `bool`

Upcasting(to a supertype) are always allowed

In [23]:
```
print(int(True))
print(float(1))
print(str(1.0))
```

```
1
1.0
1.0
```

While downcasting(to a subtype) needs type check and our attention

In [24]:
```
print(float('1.0'))
print(int('1'))
print(bool('False')) # ?
```

```
1.0
1
True
```

Now it's time to look back at `input` function, AGAIN

In [25]:
```
var_from_input = input('Please input your favorite number:')
print(type(var_from_input))
print(isinstance(var_from_input, int))
```

```
Please input your favorite number:5566
<class 'str'>
False
```

`input` function always stores our input as string

We can then convert strings to our desired data type if specific computing follows.

In [26]:
```
var_from_input = int(input('Please input your favorite number:'))
print(type(var_from_input))
print(isinstance(var_from_input, int))
print(var_from_input/100)
```

```
Please input your favorite number:5566
<class 'int'>
True
55.66
```