Introduction to Computer Programming

Objects, Variables and Operators

Welcome!

Terms of engagement

- Be respectful of each other (be aware there are different levels of experience in the room)
- Be helpful to each other (explaining a concept to someone else is a great way to reinforce your own understanding)
- Watch the videos before coming to the in-person lecture each week
- Ask questions
- Attend
  - in-person lecture
  - lab
  - lab support session
- Sit in the same seat to help the TAs monitor your progress

You may want to open the video slides now in Google Colab using the link on Blackboard

Useful for:

- Q&A
- group exercises

Video Q&A

Make use of the discussion board

Add any questions about the pre-watch/pre-read material to the Course Discussion Board.

Blackboard page >> Course tools >> Discussion Board >> EMAT10007_2023_TB-1 >> Ask a question

We will answer questions at beginning of the next lecture or by posting a response.

The deadline for posting a question to recieve a response by, or in, the next lecture is the day before the lecture (Monday), 9am.

Objects

Objects

- Every item of data (numbers, text characters etc) in a Python program can be described by the term **object**
- The type of an object determines what properties it has and how it can be used in the Python program

In [2]:
```
30
'Python'
1.2
```

Out[2]:
```
1.2
```

Variables and Variable Assignment

- A variable is a name that refers or points to a particular object
- By *assigning* an object to a variable, we allow it to be manipulated within the program, using the variable name
- To create a variable, we simply assign it a value
- Assignment is achieved with a single equals sign (=)

In [17]:
```
b = 4

print(b)
```

```
4
```

The Python function `print()` *displays whatever is between the parentheses* `(...)`

Type Conversion

The value assigned a variable can be specified or converted by *casting*

Constructor functions are used for casting.
The function name represents the desired data type.
The variable to be cast is included between the parentheses `()`

- `int()` - constructs an integer (e.g. from a float (by removing all decimals), or a string (the string must represent a whole number)
- `float()` - constructs a float (e.g. from an integer, or a string (the string must represent a float or an integer)
- `str()` - constructs a string (e.g. from an integer, float or Boolean)
- `bool()` - constructs a Boolean (e.g. from an integer, float or string)

A full list of possible type conversions can be found under 'Type Conversion' (https://realpython.com/python-data-types/#type-conversion (https://realpython.com/python-data-types/#type-conversion))

Example

Convert a float to a) integer b) string

In [15]:
```
a = 1.2

b = int(a)
print(b)

c = str(a)
print(c)
```

```
1
1.2
```

Example

Convert a string to a) integer b) float

In [23]:
```
a = '1'    # Note float string e.g. 1.2 can't convert to int
b = int(a)
print(b)


a = '1.2'
b = float(a)
print(b)
```

```
1
1.2
```

Example

Convert from an integer to a) float b) string

In [25]:
```
a = 1

b = float(a)
print(b)

c = str(a)
print(c)
```

```
1.0
1
```

Example

Convert a) integer b) float c) string, to Boolean

In [26]:
```
a = 1
b = 1.0
c = '1'

d = bool(a)

print(d)
```

```
True
```

Boolean representations of other object types

Outcomes when casting other object types as Boolean values:

Object type	`True`	`False`
`int`	non-zero	zero (`0`)
`float`	non-zero	zero (`0.0`)
`string`	non-empty string	empty string (`''`)

Operators (in order of precedence)

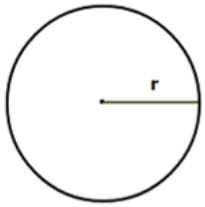
1. Parentheses
2. Arithmetic operators (top to bottom)
   - `**` Exponent
   - `/` , `*` , `//` , `\%` Division, multiplication, floor division, modulo (evaluated left to right)
   - `+` , `-` Addition, subtraction (evaluated left to right)
3. Comparison operators: `<` , `<=` , `>` , `>=` , `!=` , `==` (evaluated left to right)
4. Assignment operators `=` , `/=` , `*=` , `//=` , `\%=` , `+=` , `-=` ....
5. Identity operators `is` , `is not`
6. Logical `not`
7. Logical `and`
8. Logical `or`

Example: Arithmetic Operators

Find the area, $A = 28.278$ m^2, of a circle with

$A = \pi r^2$

where:
$\pi = 3.142$
radius, r = 3 metres



In [1]:
```
pi = 3.142
r = 3

# Calculate the area of the circle
A = pi * r**2

print(A, 'm2')
```

```
28.278 m2
```

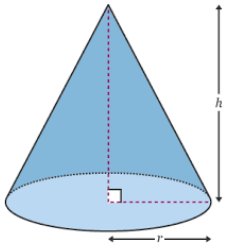
Example: Arithmetic Operators

Work as a table to write out the following operation using Python arithmetic operators

Find the volume, $V = 41.89\dot{3}$ cm ^{3}, of a cone

$V = \frac{Ah}{3}$

where:
$\pi = 3.142$
height, h = 10 cm
base radius, r = 2 cm
base area, $A = \pi r^2$



In [ ]:
```
```

What does the output look like if `h` and `r` are in units m, and the output is in units m^3?

```
h = 0.1
r = 0.02
```

Scientific notation

The `e` and `E` characters can also be used for scientific notation

n `e` m represents $n \times 10^m$

In [15]:
```
h = 0.0000000001
h = 1 * 10**-10
h = 1e-10
h = 1E-10
print(h)
```

```
1e-10
```

Comments

A line of code that begins with the hash `#` symbol is a **comment**.

Comments are used for two things:

1. A line in the code that we don't want to run.
2. A human-readable annotation in the code.

```
r = 3        # radius
pi = 3.142

# Calculate the area of the circle
A = pi * r**2
```

Let's go back an add comments to the previous examples

Example: Arithmetic Operators

Create a variable `a` and assign it a `string` value with two or more characters.

Create a variable `b` and assign it a `string` value with three or more characters.

Use an arithmetic operator to connect the **first two** letters of `a` and the **last three** letters of `b` to make a new string.

In [47]:
```
a = 'hello'

b = 'world'

c = 'hello' + 'world'

print(c)
```

```
helloworld
```

Example: Comparison Operators

Check if the `temperature` is lower than the `threshold`

```
temperature = 30
threshold = 25
```

In [3]:
```
temperature = 30
threshold = 25

print(temperature < threshold)
```

```
False
```

Example: Comparison Operators

Work as a table to write out the following operation using Python comparison operators

Check if the student score is greater than or equal to the pass mark, 40

```
score = 30
pass_mark = 40
```

In [ ]:
```

```

Example: Comparison Operators

Work as a table to write out the following operation using Python comparison operators

Check if the `student_name` matches the value of `name`

```
student_name = 'Tim'
name = 'tim'
```

In [ ]:
```

```

Floating point error and comparison operators

Care must be taken when comparing floating point values.

In [9]:
```
1.2 - 1 == 0.2
```

Out[9]:
```
False
```

Why does the expression evaluate to `False` ?

This is due to error in floating point number storage.

When we compute `1.2 - 1`, any error in the stored values of `1.2` and `1` compounds.

This compounded error may be different from when `0.2` is defined explicitly.

In [17]:
```
format(0.2, '.17f')
```

Out[17]:
```
'0.20000000000000001'
```

In [57]:
```
format(1.2, '.17f')
```

Out[57]:
```
'1.19999999999999996'
```

In [56]:
```
format(1, '.17f')
```

Out[56]:
```
'1.00000000000000000'
```

In [58]:
```
format(1.2 - 1, '.17f')
```

Out[58]:
```
'0.19999999999999996'
```

In [13]:
```
a = 1.2 - 1
b = 0.2
a < b
```

Out[13]:
```
True
```

So while we can use comparison operators on **explicitly defined** values...

In [31]:
```
a = 0.2
b = 0.2
a == b
```

Out[31]:
```
True
```

...we need to take extra care when comparing floating point values that are the result of **arithmetic operations**

In [10]:
```
a = 3.0 - 2.8
b = 1.2 - 1
a == b
```

Out[10]:
```
False
```

Comparing floating point values

To test if actual value of two floating-point values are equal we can test if the difference between the two numbers is within some tolerance.

The Python function `abs()` returns absolute value of the value within the parentheses

In [1]:
```
tolerance = 1e-10
print(tolerance)
```

```
1e-10
```

In [45]:
```
a = 1.2 - 1
b = 0.2

abs(a - b) < tolerance # Equivalent to a == b
```

Out[45]:
```
True
```

Standard comparison	Floating point comparison
`x == y`	`abs(x - y) < tolerance`
`x != y`	`abs(x - y) > tolerance`
`x > y`	`x - y > tolerance`
`x < y`	`x - y < tolerance`
`x >= y`	`abs(x - y) < tolerance or x - y > tolerance`
`x <= y`	`abs(x - y) < tolerance or x - y < tolerance`

Alternative approach:

Select a degree of accuracy within which to compare the numbers.

Round the numbers to the required degree of accuracy.

The Python function `round()` rounds a number to a specified number of decimal places.

In [12]:
```
dp = 10

a = 1.2 - 1
b = 0.2

a = round(a, dp)
b = round(b, dp)

print(a < b)
print(a == b)
```

Out[12]:
```
False
```

The Python function `round()` rounds the first comma-seperated value within the parentheses to the number of decimal places represented by the second comma-seperated value within the parentheses

Logical operators

```
x and y
```

Process:
Return `x` if the **Boolean** value of `x` is `False` ; otherwise, return `y` .

Output if `x` and `y` are both Booleans:
`True` if `x` **and** `y` are both `True` .

```
x or y
```

Process:
Return `x` if the **Boolean** value of `x` is `True` ; otherwise, return `y` .

Output if `x` and `y` are both Booleans:
`True` if at least one of `x` , `y` has the value `True` .
Otherwise `False` .

Example: Logical Operators

Test if `a` is equal to `b` **and** `c` is equal to `d`

```
a = 1
b = 2
c = 1.5
d = 1.5
```

In [68]:
```
a = 1
b = 2
c = 1.5
d = 1.5

a == b and c == d
```

Out[68]:
```
False
```

Example: Logical Operators

Test if comparisons `e` and `f` are `True` but `g` is `False`

```
e = 1 < 2
f = 2 == 2.0
g = 3 >= 1
```

In [5]:
```
e = 1 < 2
f = 2 == 2.0
g = 3 >= 1

e and f and not g
```

Out[5]:
```
False
```

Example: Logical Operators

Work as a table to write out the following operation using Python logical operators

Test if comparisons `h`, `i` and `j` are all `True`

```
h = 1 < 2
i = 2 == 2.0
j = 3 >= 1
```

In [ ]:

Example: Logical Operators

Check if `k` is less than `n` **or** `m` is less than `n`

```
k = 1
m = 2
n = 1.5
```

In [8]:
```
k = 1
m = 2
n = 1.5

k < n or m < n
```

Out[8]: `True`

This is not the same as:

In [71]:
```
k or m < n
```

Out[71]: `1`

```
x or y
```

Process:
Return `x` if the **Boolean** value of `x` is `True`; otherwise, return `y`.

```
k or m < n
```

"Return `k` if the **Boolean** value of `k` is `True`; otherwise, return `m < n`."

The Boolean value of `k` (`k` has the value `1`) is `True`, therefore `k` is returned

Example: Logical Operators

Check if `k` is less than `n` **and** `m` is less than `n`

```
k = 1
m = 2
n = 1.5
```

In [6]:
```
k = 1
m = 2
n = 1.5

k < n and m < n
```

Out[6]:
```
False
```

This is not the same as:

In [7]:
```
k and m > n
```

Out[7]:
```
True
```

```
x and y
```

Process:
Return `x` if the **Boolean** value of `x` is `False` ; otherwise, return `y` .

```
k and m > n
```

Return `k` if the **Boolean** value of `k` is `False` ; otherwise, return `m > n`

The Boolean value of `k` (`k` has the value `1`) is `True` , therefore `m > n` is returned (`m > n` has the value `True`)

Chained Comparison Operators

Comparison operators can be chained together if they feature the same operand

The following 2 statements are equivalent

```
x < y < z
```

```
x < y and y < z
```

Example: Chained Comparison Operators

Check if the temperature of an aquarium is within the allowable range 24 – 26.5°C

```
    temp = 30
    temp_low = 24
```

In [55]:
```
temp = 30
temp_low = 24
temp_high = 26.5

# temp_low <= temp and temp <= temp_high

temp_low <= temp <= temp_high
```

Out[55]:
```
False
```

Example: Chained Comparison Operators

Check that the value of variables `a` to `d` increase in alphabetical order

In [84]:
```
a = 1
b = 2
c = 3
d = 4

# a < b and b < c and c < d

a < b < c < d
```

Out[84]:
```
True
```

Summary

- Every **object** has a type (`int` , `float` , `string` ....).
- A **variable** is a name that refers or points to a particular object
- **Arithmetic operators** (+, -, /, * ....)
  Used with numeric values to perform mathematical operations (behave differently with strings).
- **Comparison operators** (==, !=, <, > ....)
  Compare two *operands*.
  Output is a *Boolean* (True or False) value.
  Comparison operators can be stacked e.g. `x < y <= z`
- **Identity operators** (`is` , `is not` ....)
  Checks if two *operands* are identical.
  Outout is a *Boolean* (True or False) value.
- **Logical operators** (`and` , `or`)
  Compare Boolean `True` or `False` *operands* (e.g. outcomes of two *comparison operations*) to form logic statements.
  Outout is a *Boolean* (True or False) value.
  Logical `not` operator returns the inverse Boolean value of an operand.
- **Assignment operators** (+=, -=, /= ....)
  Reassign the value of a variable.

Need to see some more examples?

Objects and Variables
https://www.w3schools.com/python/python_variables.asp (https://www.w3schools.com/python/python_variables.asp)
https://www.geeksforgeeks.org/python-variables/ (https://www.geeksforgeeks.org/python-variables/)

Operators
https://www.w3schools.com/python/python_operators.asp (https://www.w3schools.com/python/python_operators.asp)
https://www.geeksforgeeks.org/python-operators/ (https://www.geeksforgeeks.org/python-operators/)
https://www.programiz.com/python-programming/operators (https://www.programiz.com/python-programming/operators)
https://pynative.com/python-operators/ (https://pynative.com/python-operators/)

Want to take a quiz?

Objects and Variables
https://realpython.com/quizzes/python-variables/ (https://realpython.com/quizzes/python-variables/)
https://pynative.com/python-variables-and-data-types-quiz/ (https://pynative.com/python-variables-and-data-types-quiz/)

Operators
https://realpython.com/quizzes/python-operators-expressions/ (https://realpython.com/quizzes/python-operators-expressions/)
https://pynative.com/python-operators-and-expression-quiz/ (https://pynative.com/python-operators-and-expression-quiz/)

Want some more advanced information?

Objects and Variables
https://realpython.com/python-data-types/ (https://realpython.com/python-data-types/)
https://pynative.com/python-variables/ (https://pynative.com/python-variables/)

Operators
https://realpython.com/python-operators-expressions/ (https://realpython.com/python-operators-expressions/)

In [ ]: