Software Development 1

Data Types , Variables and Operators

TUDublin –Tallaght Campus

E.Costelloe

Data

- We need to store data in our programs, e.g. student names, results, course names, etc..
- In Python, data takes the form of objects
- Objects are essentially just pieces of memory, with values and sets of associated operations.
- As we'll see, everything is an object in a Python script. Even simple numbers qualify, with values (e.g., 99)
- Python has built-in object types, as seen below in the table,
- Numbers and strings represent numeric and textual values, respectively,

Object type	Example literals
Numbers	1234, 3.14
Strings	"John Smith", "Python"

- Ref: Learning Python Lutz M.

Variables

- Variables in Python are simply names, created by you(or Python) that are used to keep track of information in your program, they are reserved memory locations that store values.
- Variables are created when they are first assigned values.
- Variables are replaced with their values when used in expressions.
- Variables must be assigned before they can be used in expressions.
- Variables refer to objects and are never declared ahead of time.
- E.g
- a = 3
- b = 7
- Cause the variables **a** and **b** to be created automatically
- The Interpreter is responsible for creating the actual memory location, and assigning it when required or termination of the variable when no longer required.
- Some languages require you to specify the variable type and is unchangeable during execution, this takes time and forward planning whereas Python:

Will assign the type automatically.

Change the type during executing if required.

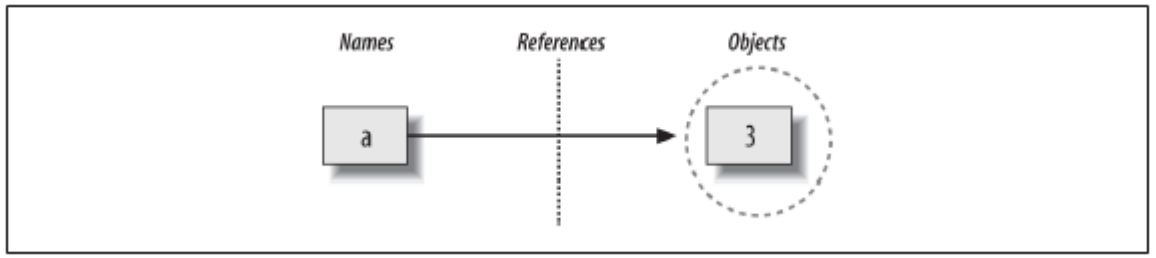
Creating a variable in Python

When we say the following, i.e. assign a value to a variable

E.g. a = 3

Python will perform three distinct steps to do this:

1. Create an object to represent the value **3**
2. Create the variable **a**, if it does not yet exist
3. Link the variable **a** to the new object **3**



Whenever the variable **a** is later used Python will retrieve the value it links to (references)i.e. 3

(As optimization, Python caches and reuses certain kinds of unchangeable objects, such as small integers and strings, implementation dependent 3.x , normally -5 .. 256 cached)

Variables

- Naming variables is an important task, saving time in the long run.

  (remember the area algorithm that we created earlier?)

- We used variables to record and store certain information about the
  - *length*
  - *width*
  - *area*
- Naming conventions are important. (ref PEP*)

- Programs can have many variables, so do not just name them var1, var2 etc..

Variables

Naming conventions:

Can contain only digits(i.e. 0..9), letters and underscores

- Must start with a letter(or _) use the letter for the moment as starting with _ have special meaning(pseudo private- internal use)
- Must not start with a number (but it may contain one)
- No spaces
- No special characters (such as '@')
- Not a keyword
- Python conventions for variable naming;
  - usually use all lower case letters, joined by underscores
- Not too long !
- Case sensitive! X is not the same as x – two different variables

Variables

Python Keywords:

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Variables Naming Examples

Correct naming examples:

i_am_a_variable

item_27

class_7

more_at

Incorrect naming examples:

I am a variable

27_item

class

more@

Data Types

- Most programming languages have very specific types.
  - *Even whole numbers can have several types (int, long, short etc..)*

- Python types:

Integer	***int***	Positive and Negative whole numbers, no fractional part, Python allows integers to have unlimited precision—they can grow to have as many digits as your memory space allows.
Floating point	***float***	Positive and Negative floating point numbers, they have a fractional part
String	***str***	Sentences, collection of characters
Boolean	***bool***	True or False

Data Types

- In Python you do not have to predefine the data types when declaring and initialising a variable:
- Integers are written as a sequence of decimal digits. A **decimal** is any number in our base-ten number system Floating-point numbers have a decimal point
- If you write a number with a decimal point, Python makes it a floating-point object and uses floating-point (not integer) math when the object is used in an expression

```
x = 5
y = 3.14
z = "PI"
ok = True
```

- The above example declares and initialises 4 variables:
  - *Integer*
  - *Floating point*
  - *String – when encased in quotes*
  - *Boolean – values of **True** or **False** only*

Variables

- We can also print out the variable's type using a Python built in function (**type(variablename)**):

```
x = 5
y = 3.14
z = "PI"
ok = True


print(type(x))
print(type(y))
print(type(z))
print(type(ok))
```

```
Output:
    <class 'int'>
    <class 'float'>
    <class 'str'>
    <class 'bool'>
```

What is the output?

```
print(type(10))
print(type("10"))
print(type(10.5))
print(type(False))
```

Variables

- The Python Interpreter can also change the type during run time, this is not the case in most other languages, where they would throw an exception(i.e. Error):

```
x = 5
print(type(x))
x = 5.1
print(type(x))
```

```
Output:
    <class 'int'>
    <class 'float'>
```

- We did not have to tell the Interpreter that ***x*** (currently an int) needs to be changed to a float.

- What is happening is that we are making **x** point to (reference) different types of objects(types are associated with objects)

Variables

- We can specify the variable to be a float (Cast it(i.e. convert it)to a float):

```
x = float(5)
print(type(x))
x = 5.1
print(type(x))
```

```
Output:
    <class 'float'>
    <class 'float'>
```

Casting

int(x) – converts x to an int, but doesn't round, it chops off the fractional part if converting a floating point number

```
x = int("32")
print(x)
x = int(3.999)
print(x)
x = int(-2.3)
print(x)
x = int("Hello")
print(x)
```

```
32
3
-2
Traceback (most recent call last):
  File "C:/Users/Eileen/PycharmProjects/week2varaiblesnotes/variablesnotes.py", line 187, in <module>
    x = int("Hello")
ValueError: invalid literal for int() with base 10: 'Hello'
```

float(x) – converts x to a float

str(x) - converts x to a string

bool(x) - converts x to a boolean (if x numeric and 0 result is False, otherwise result will be True, even if x negative)

```
x = bool(0)
print(x)
x = bool(234)
print(x)
```

```
False
True
```

Variables

We can also cast from a float to an int:

```
x = float(5)
print(type(x))
x = int(5.1)
print(x)
print(type(x))
```

Output:

```
<class 'float'>
<class 'int'>
```

- what is the printed value of x?
- What would happen if we set x to be 5.7 and then cast it to an integer? Print the type and value of x to the screen.
- **Write the code in Python.**

Variables

- Answer:

```
x = int(5.7)
print(type(x))
print(x)
```

- Output:

```
<class 'int'>
5
```

Exercises

```
x = int("456")
print(x)
x = int(9.25)
print(x)
x = int(-77.56)
print(x)
```

What is the output ?

```
456
9
-77
```

Variables

- We can cast any Python variable:

```
my_int = int("5")
my_float = float("5.5")
my_string = str(5)
my_bool = bool("False")

print(type(my_int))
print(type(my_float))
print(type(my_string))
print(type(my_bool))
```

Output:

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

Variables

Be careful:

```
my_int = int("5")
my_float = float("5.5")
my_string = str(5)
my_bool = bool("False")

print(my_int)
print(my_float)
print(my_string)
print(my bool)
```

Output:

```
5
5.5
5
True
```

All strings evaluate to "True" unless they are empty; which they then evaluate to "False".

Variables

- We will look a strings and Boolean in more detail later, but for now lets look at printing variables and using operators on them.
- Earlier we looked at printing numbers with strings, we can do the same with variables:

```
x = 5
print("The answer is:", x)
print("The answer is: " + str(x))
print("The answer is: {0}".format(x))
```

```
Output:
    The answer is: 5
    The answer is: 5
    The answer is: 5
```

Variables

- If we want to print two variables with some descriptive text (a string).

```
x = 5
y = 7
print("The answer is:", x, y)
print("The answer is: " + str(x) + " " + str(y))
print("The answer is: {0} {1}".format(x, y))
```

```
Output:
    The answer is: 5 7
    The answer is: 5 7
    The answer is: 5 7
```

Variables

- If we want to print two variables with some descriptive text (a string).

```
print("The answer is:", x, y)
print("The answer is: " + str(x) + " " + str(y))
print("The answer is: {0} {1}".format(x, y))
```

- For this course, we will be using the first method to print strings and variables. Each has its own merits, and later you may need to use one method over another, but for now lets use one convention.

Variables

- Why can I not just use the plus symbol to print two variables?

```
x = 5
y = 7
print("The answer is:", x + y)
```

```
Output:
    The answer is: 12
```

- Generally it is good practice to avoid variable operations in print statements.
  - *It can be difficult to read*
  - *You might need to use the value later, thus re-computing the statement*

Numeric Literals

Integers are positive and negative whole numbers

They are written as sequences of decimal digits

e.g.

a = 10

Floating point numbers are positive and negative numbers with a fractional part and are written with a decimal point.

If you write a number with a decimal point Python makes it a floating point object and uses floating point math when using it in an expression.

e.g.

b = 1.345

Operators (Part 1)

Now that we can create and print variables, lets look at operators , mathematical operators first:

```
x = 10
y = 7
answer = x + y          # Addition
answer = x - y          # Subtraction
answer = x * y          # Multiplication
answer = x / y          # Division (Always results in float)
answer = x // y         # Floor Division (Results in whole number)
answer = x % y          # Modulus (remainder after division)
answer = x ** y         # Exponent (x to the power of y)
```

= is the assignment operator

NB The variable to the left of the equals is the only variable modified

Note: with // if one number is float it will yield a float , always the net effect of // is to round down

```
print(11.0 // 2)
```

- Will yield 5.0

Modulus % Exercises

result = 29 % 9

result = 6 % 8

result = 40 % 40

result = 10 % 2

result = 21.0 % 5

Operators

- What do you think will happen here?

```
x = 5
x = x + 2
print(x)
```

- The variable to the left of the equals sign(assignment operator) is the only variable modified

Operators

- What do you think will happen here?

```
x = 5
x = x + 2
print(x)
```

Output:
7

- Generally the right hand side of the equals sign is calculated first by the Interpreter.
- Lets step through this statement:

Operators

```
x = 5
x = x + 2
print(x)
```

- Line 1 :the variable x is created, and the value 5 is assigned to this variable.

- Line 2: On the right hand side of the statement:
  - *The value is retrieved from the variable x (5), this is then added to 2 resulting in 7.*
- Line 2: on the left hand side of the statement:
  - *The value 7 is now inserted into the variable x*

- Line 3: The value is retrieved from x and printed to the screen.

Operators

- This situation happens a lot in programming, for example to add one to a number, that is to, increment it.

```
x = 5
x = x + 1
print(x)
```

```
Output:
    6
```

- There is a shorter way to achieve this statement.

Operators

- Second option

```
x = 5
x += 1
print(x)
```

Output:

6

Operators (part 2)- Augmented Assignment Operators

Statement	Code example	Equivalent to
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x – 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5

Operators (Order of Precedence)
BODMAS

Order	Operator	Description
1	( )	Parentheses
2	**	Exponentiation
3	*, /, %	Multiplication, Division, Remainder
4	+, -	Addition, Subtraction

- If there is equal order, the Interpreter works left to right.
- Therefore it is reasonable to assume that brackets should be used to confirm your mathematical approach.

Operators

Example, evaluate the following: $\frac{4+5}{1+2}$

- A

```
answer = 4 + 5 / 1 + 2
print(answer)
```

- B

```
answer = (4 + 5) / (1 + 2)
print(answer)
```

Operators

Example, evaluate the following: $\frac{4+5}{1+2}$

- A

```
answer = 4 + 5 / 1 + 2
print(answer)
```

Output:
11

- B

```
answer = (4 + 5) / (1 + 2)
print(answer)
```

Output:
3

- Use BODMAS to avoid such issues.

Mixing data types in expressions

You may include different data types in an expression e.g. you can add an integer and a floating point number

40 + 3.14

What type is the result, integer or floating point?

Answer: in mixed type **numeric** expressions, Python converts operands up to the type of the most complicated operand and then performs the math on same type operands.

In Python integers are simpler than floating point.

So in the above example the integer 40 is converted to a floating point and floating point math yields the result of 43.14

In general, Python does not convert across other type boundaries automatically, e.g. adding a string and an integer will result in an error, unless you manually convert one or the other

Exercises

Evaluate:

3 + 5 // 7

3 * 3 + 3 % 2

3 + 2 // 5 + -2 * 4

Operators: In Class

Example, evaluate the following assuming that PI is 3.14, and the radius of a sphere is 7.

Find the volume of the sphere given: Volume = $\frac{4}{3}\ \pi\ r^3$

Hint: Try use variables to store data when you can

Operators: In Class

Example, evaluate the following assuming that PI is 3.14, and the radius of a sphere is 7.

Find the volume of the sphere given: Volume = $\frac{4}{3}\ \pi\ r^3$

```
pi = 3.14
radius = 7
answer = ((4 / 3) * pi) * radius ** 3
print(answer)
```

Output:

1436.0266666666666

Hint: Always check your work with a calculator

Printing results:

If the following example was to be monetary, for example an insurance quote, the large amount of decimals would possible confuse a user.

Lets round this to 2 decimal places:

```
pi = 3.14
radius = 7
answer = ((4 / 3) * pi) * radius ** 3
answer = round(answer, 2)
print(answer)
```

Output:

1436.03

The round function takes two arguments:

Printing results:

- round(): function

Stores the result in answer

```
answer = round(answer, 2)
```

The variable to round

The number of places to round it to.

Rounds up or down as the case may be

```
print(round(12.3))
print(round(12.6))
```

```
12
13
```

If two numbers are equally close rounding is done towards the even choice

```
print(round(12.5))
print(round(13.5))
```

```
12
14
```

Data Type: Strings

- Strings are used to record textual information(name, address etc.), they are immutable in Python.
- This means they cannot be modified once created.

- Consider them as a sequence – i.e. a positionally ordered collection of characters, maintaining a left to right order among the items they contain; strictly speaking they are a sequence of one-character strings

```
my_string = "hello world"
print(my_string)
```

- There are some operators that work on strings, but not as many as numeric variables.

Variables: Strings

- Strings can be added together (concatenation) with the + operator

```
string1 = "hello"
string2 = "world"
answer = string1 + string2
print(answer)
```

Output:

helloworld

Variables: Strings

- Strings can be added together (concatenation)

```
string1 = "hello"
string2 = "world"
answer = string1 + " " + string2
print(answer)
```

```
Output:
    hello world
```

Variables: Strings

- Strings can multiplied with the * operator:

```
string1 = "hello"
answer = string1 * 3
print(answer)
```

Output:

hellohellohello

- Write the code to print hello 3 times but place a space between the hello strings, ensure there is no trailing space, i.e. at the end of the last hello string

Exercise

Write the code that would create two variables, number1 and number2, assign 100.0 and 200.0 to those variables respectively, divide number1 by number2 storing the result in answer and print the answer to the screen, to 1 decimal place: Output as below

```
Result of dividing 100.0 by 200.0 is: 0.5
```

```
number1 = 100.0
number2 = 200.0
answer = number1 / number2
# print("Result of dividing", number1 , "by", number2, "is:",round(answer, 1))
print("Result of dividing", number1, "by", number2, "is:", answer)
```

Exercise

You have 1729 cents stored in a jar. Write a program that will determine how many Euros and cents are in the jar.

Output:

```
The number of euros is 17
The number of remaining cents from 1729 is 29
```

```
CENTS_PER_EURO = 100
cents = 1729
```

```
euros = cents // CENTS_PER_EURO
rem_cents = cents % CENTS_PER_EURO
```

```
print("The number of euros is", euros)
print("The number of remaining cents from",cents,"is", rem_cents)
```