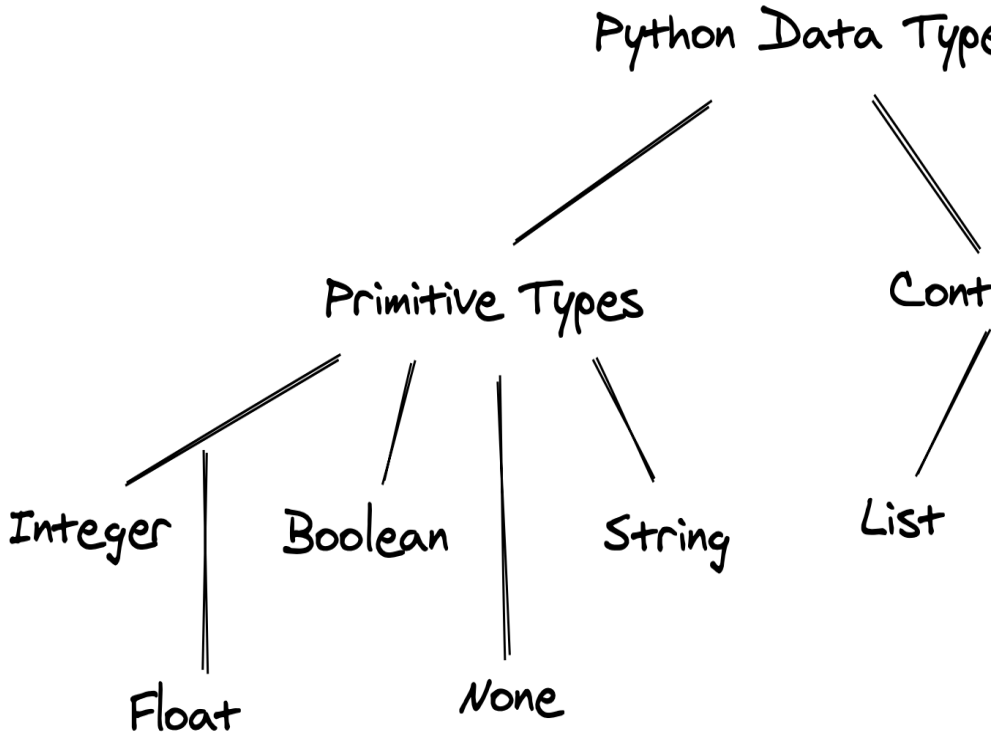
2 Variables, expressions and statements

2.1 Data types

Python has several built-in data types for storing different kinds of information in variables.



2.1.1 Primitive

Integer, float, boolean, None, and string are *primitive data types* because they represent a single value.

2.1.2 Containers

Other data types like list, tuple, and dictionary are often called *data structures* or *containers* because they hold multiple pieces of data together. We'll discuss these datatypes in chapter 5.

The data type of the object can be identified using the in-built python function

type(). For example, see the following objects and their types:

```
type(4)
```

```
int
```

```
type(4.4)
```

```
float
```

```
type('4')
```

```
str
```

```
type(True)
```

```
bool
```

2.1.3 Practice exercise 1

What is the datatype of the following objects?

1. 'This is False'
2. "This is a number"
3. 1000
4. 65.65
5. False

2.2 Constants and Variables

A constant is a value that cannot be changed. It may be a number, string or any other datatype. Below are some examples of printing a constant:

```
print(4)
```

```
4
```

```
print("This is a string and also a constant")
```

```
This is a string and also a constant
```

```
print(False)
```

```
False
```

A variable is an object whose value can be changed. For example, consider the object below:

```
x = 2
```

In the above code the variable `x` has been assigned a value of `2`. However, the value of `x` can be changed:

```
x = 3
print("x =", x)
```

```
x = 3
```

Thus, the object `x` in the above code is a variable that refers to a memory location storing the constant value of 3.

2.2.1 Variable names

There are a some rules for naming variables:

1. A variable name must start with a letter or underscore _
2. A variable name may consist of letters, numbers, and underscores only

For example, some of the valid variable names are `salary`, `text10`, `_varname`. Some of the invalid variable names are `salary%`, `10text`, `varname)`.

3. Variable names are case-sensitive. For example, the variable `Varname` will be different from `varname`.
4. There are certain *reserved words* in python that have some meaning, and cannot be used as variable names. These reserved words are:

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

Best coding practice: Variables should be named such that they are informative of the value they are storing. For example, suppose we wish to compute the income tax a person has to pay based on their income and tax rate. Below are two ways of naming variables to do this computation:

```
income = 80000
tax_rate = 0.15
print("Income tax = ", income*tax_rate)
```

```
Income tax =  12000.0
```

```
a = 80000
b = 0.15
print("Income tax = ",a*b)
```

```
Income tax =  12000.0
```

The former code chunk is better than the latter one as it makes the code easy to read and understand.

Python style guide: Please refer to the python style guide for best coding practices, such as naming variables, using spaces, tabs, and styling the different components of your code.

2.2.2 Practice exercise 2

2.2.2.1 Variables or constants?

In the statements below, classify the objects as variables or constants?

1. value = “name”
2. constant = 7
3. another_const = “variable”

4. True_False = True

2.2.2.2 Valid variable names?

Which of the following variable names are valid?

1. var.name
2. var9name
3. _varname
4. varname*

2.3 Assignment statements

Values are assigned to variables with the assignment statement (=). An assignment statement may have a constant or an expression on the right hand side of the (=) sign, and a variable name on the left hand side.

For example, the code lines below are assignment statements

```
var = 2
var = var + 3
```

2.4 Expressions

The mathematical operations and their corresponding operators are as follows:

1. Exponent: **
2. Remainder: %
3. Multiplication: *
4. Division: /
5. Addition: +
6. Subtraction: -

The operators above are in decreasing order of precedence, i.e., an exponent will be evaluated before a remainder, a remainder will be evaluated before a multiplication, and so on.

For example, check the precedence of operators in the computation of the following expression:

```
2+3%4*2
```

```
8
```

In case an expression becomes too complicated, use of parenthesis may help clarify the precendence of operators. Parenthesis takes precendence over all the operators listed above. For example, in the expression below, the terms within parenthesis are evaluated first:

```
2+3%(4*2)
```

```
5
```

2.4.1 Practice exercise 3

Which of the following statements is an assignment statement:

1. x = 5
2. print(x)
3. type(x)
4. x + 4

What will be the result of the following expression:

```
1%2**3*2+1
```

2.5 Converting datatypes

Sometimes a value may have a datatype that is not suitable for using it. For example, consider the variable called *annual_income* in the code below:

```
annual_income = "80000"
```

Suppose we wish to divide `annual_income` by 12 to get the monthly income. We cannot use the variable `monthly_income` directly as its datatype is a string and not a number. Thus, numerical operations cannnot be performed on the variable `annual_income`.

We'll need to convert *annual_income* to an integer. For that we will use the python's in-built `int()` function:

```
annual_income = int(annual_income)
monthly_income = annual_income/12
print("monthly income = ", monthly_income)
```

```
monthly income =  6666.666666666667
```

Similarly, datatypes can be converted from one type to another using in-built python functions as shown below:

```
#Converting integer to string
str(9)
```

```
'9'
```

```
#Converting string to float
float("4.5")
```

```
4.5
```

```
#Converting bool to integer
int(True)
```

```
1
```

Sometimes, conversion of a value may not be possible. For example, it is not possible to convert the variable `greeting` defined below to a number:

```
greeting = "hello"
```

However, in some cases, mathematical operators such as `+` and `*` can be applied on strings. The operator `+` concatenates multiple strings, while the operator `*` can be used to concatenate a string to itself multiple times:

```
"Hi" + " there!"
```

```
'Hi there!'
```

```
"5" + '3'
```

```
'53'
```

```
"5"*8
```

```
'55555555'
```

2.6 User input

Python's in-built `input()` function can be used to accept an input from the user. For example, suppose we wish the user to onput their age:

```
age = input("Enter your age:")
```

```
Enter your age:34
```

The entered value is stored in the variable `age` and can be used for computation.

2.6.1 Practice exercise 4

Ask the user to input their year of birth, and print their age.

2.7 Commenting code

The `#` symbol can be used to comment the code. Anything after the `#` sign is ignored by python. Commenting a code may have several purposes, such as:

- Describe what is going to happen in a sequence of code
- Document who wrote the code or other ancillary information
- Turn off a line of code - perhaps temporarily

For example, below is code with a comment to describe the purpose of the code:

```
#Computing number of hours of lecture in this course
print("Total lecture hours of STAT201=",10*3*(5/6))
```

```
Total lecture hours of STAT201= 25.0
```

2.7.1 Practice exercise 5

Which of the following lines is a comment:

1. #this is a comment
2. ##this may be a comment
3. A comment#

2.8 Programming errors

There are 3 types of errors that can occur in a program - syntax errors, run-time errors, and semantic errors.

2.8.1 Syntax errors

Syntax errors occur if the code is written in a way that it does not comply with the rules / standards / laws of the language (python in this case). For example, suppose a values is assigned to a variable as follows:

```
9value = 2
```

The above code when executed will indicate a syntax error as it violates the rule that a variable name must not start with a number.

2.8.2 Run-time errors

Run-time errors occur when a code is syntactically correct, but there are other issues with the code such as:

- Misspelled or incorrectly capitalized variable and function names
- Attempts to perform operations (such as math operations) on data of the wrong type (ex. attempting to subtract two variables that hold string values)
- Dividing by zero
- Attempts to use a type conversion function such as `int` on a value that can't be converted to an `int`

For example, suppose a number is multipled as follws:

```
multiplication_result = x * 4
```

The above code is syntactically correct. However, it will generate an error as the variable `x` has not been defined as a number.

2.8.3 Semantic errors

Semantic errors occur when the code executes without an error being indicated

by the compiler. However, it does not work as inteded by the user. For example, consider the following code of mutiplying the number 6 by 3:

```
x = '6'
x * 3
```

```
'666'
```

If it was intended to multiply the number 6, then the variable `x` should have been defined as `x=6` so that `x` has a value of type `integer`. However, in the above code `6` is a `string` type value. When a `string` is multiplied by an integer, say *n*, it concatenates with itself *n* times.

2.8.4 Practice exercise 6

Suppose we wish to compute tax using the income and the tax rate. Identify the type of error from amongst syntax error, semantic error and run-time error in the following pieces of code.

```
income = 2000
tax = .08 * Income
print("tax on", income, "is:", tax)
```

```
income = 2000
tax = .08 x income
print("tax on", income, "is:", tax)
```

```
income = 2000
tax = .08 ** income
print("tax on", income, "is:", tax)
```

2.9 Practice exercise 7

The formula for computing final amount if one is earning compund interest is given by:

$$A = P\left(1 + \frac{r}{n}\right)^{nt},$$

where:
P = Principal amount (initial investment),
r = annual nominal interest rate,
n = number of times the interest is computed per year,

t = number of years

Write a Python program that assigns the principal amount of $10000 to variable *P*, assign to *n* the value 12, and assign to *r* the interest rate of 8%. Then have the program prompt the user for the number of years *t* that the money will be compounded for. Calculate and print the final amount after *t* years.

What is the amount if the user enters *t* as 4 years?