CSTWPY (Winter 2022)

Python for Beginners

An Intro and Variables

1 Learning Objectives

By the end of this week, students should be able to:

1. Understand the general syntax of Python
2. Define the following terms: **variables**, **strongly-typed** and **weakly-typed** languages, **dynamically** and **statically-typed** languages.
3. Understand the difference between **interpreted**, **complied** and **hybrid** languages.
4. Understand the downfalls and benefits of **Dynamically-typed** languages
5. Declare and **re-use variables**.
6. Identify errors involving variables: including `NameErrors`.

2 Variables

2.1 An Intro & Why Variables

Definition 2.1 (What are variables?)

Variables reference specific parts of memory where content is stored. The name ***variable*** implies that these objects may change depending on the type itself.

In either case, variable information is **checked** when the program gets executed by the user.

Variables in all languages, independent of Python is important to **simplify** your code as much as possible. For example,

Example 2.1 (Variables vs no variables)

Consider code block 1 with variables:

```python
NAME = 'therealgg13'
AGE = 21
isMajority = AGE > 18
friends = []

def addFriend(friend: str) -> None:
    friends.append(friend)

def formatNice() -> str:
    return f'{NAME} {AGE}'
```

Consider code block 2 with *limited* variables:

```python
def addFriend(friend: str, friends: list) -> list:
    friends.append(friend)
    return friends

def formatNice(name, age) -> str:
    return f'{name} {age}'

# calling addFriend
friends = addFriend("Miskity", [])
friends = addFriend("Miskity", friends)
```

Example 2.1 highlights the code simplicity we will have if we **properly** define variables. As your programs get more complex. Though the example demonstrated here is a relatively simple example, but imagine how you will keep track of values for over 100 lines of code across 100 different files. In that case, you have two options:

1. Rewrite the same value over 100 times
2. Create variables to store these values so you can access them later

Of course, option 2 is more viable!

2.2 Type checking of Variables

Definition 2.2 (Typed Languages)

Ways programming languages are typed affect their general syntax and how the program can run. Here are the **four** common types of programming languages:

1. **Strongly-typed languages** are defined by "fixed" types. This means at compile time, types of variables are already defined and **cannot** change.

2. **Weakly-typed languages** can have the ability to **coerce** types at runtime. For example, in javascript, this is completely valid code: `"1" + 1`. This is because at runtime, the `"1"` is type casted into an integer, then the addition is performed, which is what we call **type coercion**.

3. **Dynamically-typed languages** are languages where types are not checked until runtime (i.e., when the program is ran.)

4. **Statically-typed languages** are languages that check the type information when the program is compiled, not when the program is run.

Note: A weakly-typed language cannot be strongly typed and a **dynamically-typed** language cannot be **statically-typed**.

Though not every programming language are created equally, as we have **interpreted**, **complied** and **hybrid** languages. In general:

Definition 2.3 (Interpreted, complied and hybrid languages)

An ***interpreted language*** is a programming language which are generally interpreted, without compiling a program into machine instructions.

A ***compiled language*** is a programming language whose implementations are typically compilers. Compilers translate ***code*** to ***machine language***.

A ***hybrid language*** is a programming language that uses both ***interpretation*** and ***compilation***.

2.3 Declaring, assigning and reusing variables

Concept 2.1 (Declaring Variables in Python)

```python
sample_string = "breadgoesbrrr" # string
sample_float = 0.03 # float
sample_int = 90 # integer
sample_boolean = True # boolean (True, False)
sample_list = ["Hello Toki Club", "Hello MsKitty"]
# ... many others
```

Example 2.2 (Declaring variables in Python)

```python
x = 5
y = 'hello'
x = y
print(x) # 'hello'
```

Revisiting **Example 2.2** we can finally provide an answer to how Python runs your code.

Python is an interpreted language because there is **no** requirement to build the script before running it. Instead, Python runs the code and does all the type checking at **runtime**. This can lead to many problems and advantages compared to a compiled or hybrid language.

Advantage: Types can be reassigned to different types as shown in **Example 2.2**. Allowing this allows Python to be more flexible.On the other hand, some disadvantages of interpreted, dynamically typed languages are:

Disadvantage # 1: The speed will take a significant hit. This is because the interpreter must do twice the work: compilation and running the program - **all at runtime**

Disadvantage # 2: Unknown types. As a developer programming in Python, you either have to guess the types of variables or **manually** keep track of these types, as they might change throughout the program. Other times, these changes can be accidental. When this happens, your program fundamentally stops working. Subsequent sections explore this phenomenon further.

As part of your practice, you should find other disadvantages of **interpreted** and **dynamically-typed** languages such as Python. **You will be required to do additionally web searching**.

2.4 The Downfall of Python & Java comparison

Though **Example 2.2** demonstrates that Python variables can be reassigned, there is one major downfall with Python's dynamic type. Though we haven't learned functions, **you do not need** to understand **Example 2.3**.

Example 2.3 (The Downfall of Python)

Suppose you have the following variables defined. Notice here that the variable `x` is ***redefined*** to contain a different type.

The following code snippet uses material to be taught later. You do not need to understand this.

```
x = 4
x = [3+4] + [34]
y = x + [True]
x = [x[i] for i in range(len(x))]

def foo(x: str) -> str:
    return x + "4"
```

Interestingly, since `type(x) == list`, calling the function with `x` as a parameter would fail because you cannot add a list to a string. This behaviour is to be expected in all dynamically typed languages, as the variable name can be reassigned any time. The only time the type of the variable name is checked is at runtime.

This is in direct contrast to a language such as Java, which checks the variable type at compile time and enforces a type structure. Essentially, when a variable gets assigned a type, the variable must stay that type unless you assign another variable name. For example:

Example 2.4 (Java Example)

```
String x = ""; // only can be a string
int y = 223; // likewise, but an integer
```

2.5 Undefined variable names

Most programming languages do not let developers to retrieve a variable that has not been defined. That is, the user must declare all variable names before their retrieval.

Example 2.5 (Undefined Variable Example)

The following code snippet uses material to be taught later. You do not need to understand this.

```python
x = 4
x = [3+4] + [34]
y = x + [True]
x = [x[i] for i in range(len(x))]

def foo(x: str) -> str:
    return x + "4"

y = foo("3")
x = 2

# here we are trying to reference an undefined variable
z
```

In the case of **Example 2.5**, running it through a Python interpreter will yield a `NameError` because `z` is not defined.

Note that there are many other exceptions Python can output, some of which are user created.

3 Exercies

These exercises asks you about today's lecture content. Though no solutions will be provided here, we will discuss the solutions in lecture **and will be recorded**.

Note that some concepts present was not taught in this lecture, but it should not adversely affect your ability to do these questions!

3.1 Multiple Choice

From the following options, select the most *correct* statement that completes the following:

Python is a(n) ...

a) interpreted language

b) dynamically-typed language

c) statically-typed language

d) both **a)** and **b)**

e) all of the above

3.2 Short/Long Answer

Earlier, we explored two disadvantages of **interpreted** languages:

Disadvantage # 1: The speed will take a significant hit. This is because the interpreter must do twice the work: compilation and running the program - **all at runtime**

Disadvantage # 2: Unknown types. As a developer programming in Python, you either have to guess the types of variables or **manually** keep track of these types, as they might change throughout the program. Other times, these changes can be accidental. When this happens, your program fundamentally stops working.

Now, find at least **two** additional drawbacks of these languages.

Hint: Additional web searching is required.

3.3 Mulit-part

Recall this definition:

Definition 3.1 (Different Types of Programming languages)

Ways programming languages are typed affect their general syntax and how the program can run. Here are the **four** common types of programming languages:

1. **Strongly-typed languages** are defined by "fixed" types. This means at compile time, types of variables are already defined and **cannot** change.
2. **Weakly-typed languages** can have the ability to **coerce** types at runtime. For example, in javascript, this is completely valid code: `"1" + 1`. This is because at runtime, the `"1"` is type casted into an integer, then the addition is performed, which is what we call **type coercion**.
3. **Dynamically-typed languages** are languages where types are not checked until runtime (i.e., when the program is ran.)
4. **Statically-typed languages** are languages that check the type information when the program is compiled, not when the program is run.

Note: A weakly-typed language cannot be strongly typed and a **dynamically-typed** language cannot be **statically-typed**.

3.3.1 Part a

Find at least **one** language that fit into the following categories:

1. **Strongly-typed & dynamically-typed languages**
2. **Strongly-typed & statically-typed languages**

3.3.2 Part b

Explain why this is a true statement, based on your understanding on types:

Note: A weakly-typed language cannot be strongly typed and a **dynamically-typed** language cannot be **statically-typed**.

Hint: Additional web searching is required.

3.3.3 Part c - True/False

Determine if the following statement is **TRUE** or **FALSE**. If it's true, **provide a justification**. If it's false, **provide a counter example**.

All interpreted languages are **dynamically-typed**.

Hint: Additional web searching is required.

3.4 Code Snippets

Consider the following code snippets and determine the output of the `print` statements. Write **No output** if it doesn't output anything. If it returns error, state the error and **explain why**.

3.4.1

```python
x = [x + 5]
y = 2 + 23 + x + ["Toki"]
print(y)
```

3.4.2

```python
x = x
y = 3 + 4
print(h)
```

3.4.3

```python
y = 23
x = 2
z = x + 3 + y

# ADVANCED MATERIAL -- U DO NOT NEED TO UNDERSTAND
def foo(x: list) -> int:
    return sum(x)

print(foo(y))
```

3.5 Reaching Beyond the Lesson

Explain, in **your own words** why Python is the most common programming language:

1. for beginners
2. for data scientists
3. artificial intelligence

Hint: Additional web searching is required.

3.6 Bug Fixing - Understanding syntax

Consider the following block of code:

```
x = 23
y = "22'
x = 23334;

def foo(x: str) ->: {
return (x+2);
}

bar(y) => {
    return y : 2
}

print(x+y+u)
```

There are a total of **eight** errors in the code above. For each error:

1. Explain the cause of the error
2. Attempt to fix them

Congratulations!

You are now one step closer to becoming an expert in Python!