Variables

We use variables to temporarily store data in computer's memory.

price = 10

rating = 4.9

course_name = '**Python for Beginners to Advance**'

is_published = True

In the above example, • price is an integer (a whole number without a decimal point)

• rating is a float (a number with a decimal point)

• course_name is a string (a sequence of characters)

• is_published is a boolean. Boolean values can be True or False.

Comments

We use comments to add notes to our code. Good comments explain the how's and whys, not what the code does. That should be reflected in the code itself. Use comments to add reminders to yourself or other developers, or also explain your assumptions and the reasons you've written code in a certain way.

```
# This is a comment, and it won't get executed.

# Our comments can be multiple lines.
```

Receiving Input

We can receive input from the user by calling the input() function.

```
birth_year = int(input('Birth year: '))
```

The **input()** function always returns data as a string. So, we're converting the result into an integer by calling the built-in **int()** function.

Strings

We can define strings using single (‘ ‘) or double (“ “) quotes.

To define a multi-line string, we surround our string with tripe quotes (“””).

We can get individual characters in a string using square brackets [].

course = ‘**Python for Beginners to Advance**’

course[0] # returns the first character

course[1] # returns the second character

course[-1] # returns the first character from the end

course[-2] # returns the second character from the end

We can slice a string using a similar notation:

course[1:5]

The above expression returns all the characters starting from the index position of 1

to 5 (but excluding 5). The result will **be ytho**

If we leave out the start index, 0 will be assumed.

If we leave out the end index, the length of the string will be assumed.

We can use formatted strings to dynamically insert values into our strings:

name = ‘Mubi

message = f’Hi, my name is {name}’

message.upper() # to convert to uppercase

message.lower() # to convert to lowercase

message.title() # to capitalize the first letter of every word

message.find(‘p’) # returns the index of the first occurrence of p

(or -1 if not found)

message.replace(‘p’, ‘q’)

To check if a string contains a character (or a sequence of characters), we use the in operator:

```
contains = 'Python' in course
```

Arithmetic Operations

```
+
-
*
/ # returns a float
// # returns an int
% # returns the remainder of division
** # exponentiation - x ** y = x to the power of y
```

Augmented assignment operator:

```
x = x + 10
x += 10
```

Operator precedence:

1. parenthesis
2. exponentiation
3. multiplication / division
4. addition / subtraction

If Statements

```
if is_hot:
 print("hot day")
elif is_cold:
 print("cold day")
else:
 print("beautiful day")
```

Logical operators:

```
if has_high_income and has_good_credit:
 ...
if has_high_income or has_good_credit:
 ...
is_day = True
is_night = not is_day
```

Comparison operators

a > b

a >= b (greater than or equal to)

a < b

a <= b

a == b (equals)

a != b (not equals)

While loops

```
i = 1
while i < 5:
 print(i)
 i += 1
```

For loops

```
for i in range(1, 5):
 print(i)
```

- **range(5):** generates 0, 1, 2, 3, 4
- **range**(1, 5): generates 1, 2, 3, 4
- **range**(1, 5, 2): generates 1, 3

Lists

```
numbers = [1, 2, 3, 4, 5]
numbers[0] # returns the first item
numbers[1] # returns the second item
numbers[-1] # returns the first item from the end
numbers[-2] # returns the second item from the end

numbers.append(6) # adds 6 to the end
numbers.insert(0, 6) # adds 6 at index position of 0
numbers.remove(6) # removes 6
numbers.pop() # removes the last item
numbers.clear() # removes all the items
numbers.index(8) # returns the index of first occurrence of 8
numbers.sort() # sorts the list
numbers.reverse() # reverses the list
numbers.copy() # returns a copy of the list
```

Tuples

They are like read-only lists. We use them to store a list of items. But once we define a tuple, we cannot add or remove items or change the existing items.

```
coordinates = (1, 2, 3)
```

We can unpack a list or a tuple into separate variables:

```
x, y, z = coordinates
```

Dictionaries

We use dictionaries to store key/value pairs.

```
customer = {
 "name": "John Smith",
 "age": 30,
 "is_verified": True
}
```

We can use strings or numbers to define keys. They should be unique. We can use any types for the values.

```
customer["name"] # returns "John Smith"
customer["type"] # throws an error
customer.get("type", "silver") # returns "silver"
customer["name"] = "new name"
```

Functions

We use functions to break up our code into small chunks. These chunks are easier to read, understand and maintain. If there are bugs, it's easier to find bugs in a small chunk than the entire program. We can also re-use these chunks.

```
def greet_user(name):
 print(f"Hi {name}")
greet_user("John")
```

Parameters are placeholders for the data we can pass to functions. Arguments are the actual values we pass.

We have two types of arguments:

- Positional arguments: their position (order) matters
- Keyword arguments: position doesn't matter - we prefix them with the parameter name.

```
# Two positional arguments
greet_user("John", "Smith")
# Keyword arguments
calculate_total(order=50, shipping=5, tax=0.1)
```

Our functions can return values. If we don't use the return statement, by default **None** is returned. None is an object that represents the absence of a value.

```
def square(number):
 return number * number
result = square(2)
print(result) # prints 4
```

Exceptions

Exceptions are errors that crash our programs. They often happen because of bad input or programming errors. It's our job to anticipate and handle these exceptions to prevent our programs from cashing.

```
try:
 age = int(input('Age: '))
 income = 20000
 risk = income / age
 print(age)
except ValueError:
 print('Not a valid number')
except ZeroDivisionError:
 print('Age cannot be 0')
```

Classes

We use classes to define new types.

```
class Point:
 def __init__(self, x, y):
 self.x = x
 self.y = y
 def move(self):
 print("move")
```

When a function is part of a class, we refer to it as a **method**.

Classes define templates or blueprints for creating objects. An object is an instance of a class. Every time we create a new instance, that instance follows the structure we define using the class.

```
point1 = Point(10, 5)
point2 = Point(2, 4)
```

__init__ is a special method called constructor. It gets called at the time of creating new objects. We use it to initialize our objects.

Inheritance

Inheritance is a technique to remove code duplication. We can create a base class to define the common methods and then have other classes inherit these methods.

```
class Mammal:
 def walk(self):
 print("walk")
class Dog(Mammal):
 def bark(self):
 print("bark")
dog = Dog()
dog.walk() # inherited from Mammal
dog.bark() # defined in Dog
```

Modules

A module is a file with some Python code. We use modules to break up our program into multiple files. This way, our code will be better organized. We won't have one gigantic file with a million lines of code in it!

There are 2 ways to import modules: we can import the entire module, or specific objects in a module.

```
# importing the entire converters module
import converters
converters.kg_to_lbs(5)
# importing one function in the converters module
from converters import kg_to_lbs
kg_to_lbs(5)
```

Packages

A package is a directory with __init__.py in it. It can contain one or more modules.

```
# importing the entire sales module
from ecommerce import sales
sales.calc_shipping()
# importing one function in the sales module
from ecommerce.sales import calc_shipping
calc_shipping()
```

Python Standard Library

Python comes with a huge library of modules for performing common tasks such as sending emails, working with date/time, generating random values, etc.

Random Module

```
import random
random.random() # returns a float between 0 to 1
random.randint(1, 6) # returns an int between 1 to 6
```

members = [‘John’, ‘Bob’, ‘Mary’]

leader = random.choice(members) # randomly picks an item

Pypi

Python Package Index (pypi.org) is a directory of Python packages published by Python developers around the world. We use **pip** to install or uninstall these packages.

pip install openpyxl

pip uninstall openpyxl