1. Variables and functions

1. Arithmetic operators

Python for Arithmetic

- Python provides operators, which are special symbols that represent computations like addition and multiplication.
- The operators +, -, * , and / perform addition, subtraction, and multiplication, as in the following examples:

```
>>> 10/2
5
>>> 10 + 2
12
>>> 10 * 10
100
```

- Exponentiation. To calculate 6^2+3

```
>>> 6**2 + 3
39

>>> 2 * 3 ** 2
18
```

- That operator `%` or modulo operator may not be very familiar to you. It does not return a percentage, but *remainder* after dividing one number from another.

```
>>> 10 % 2
0

>>> 10 % 4
2
```

2. Assignment statement and variables

Assignment statements

- An assignment statement creates a new variable and gives it value. For example,

```
>>> n = 388
>>> n
388

>>> n - 3
385

>>> message = 'Hello World!'
>>> message
Hello World!
```

Variable names

- Variable names can be as long as you like. They can contain both letters and numbers.
- But they can't begin with a number.
- It is legal to use uppercase letters, but it is conventional to use only lowercase for variable names.
- The underscore character, _ , can appear in a name. It is often used in names with multiple words, such as *your_name* or *my_score*.
- If you give a variable an illegal name, you get a syntax error:

```
>>> 388course = 'Bus Program'
  File "<stdin>", line 1
    388course = 'Bus Program'
      ^
SyntaxError: invalid decimal literal
```

- `388course` is illegal because it begins with a number

```
>>> course@ = 388
  File "<stdin>", line 1
```

```
    course@ = 388
            ^
SyntaxError: invalid syntax
```

- `course@` is illegal because it contains an illegal character, @.

```
>>> class = 'bus Program'
  File "<stdin>", line 1
    class = 'bus Program'
          ^
SyntaxError: invalid syntax
```

- Variable name `class` is illegal because `class` is one of Python's keywords.
- The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.
- Python 3 has these 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
```

3. Interactive mode and script mode

Interactive mode

- So far we have run Python in interactive mode, which means that you interact directly with the interpreter.
- In terminal, type `python` or `python3`, then `enter`. You will be present with `>>>` in the terminal window, where you can then run live, interactive code.

Script mode

- In VS code, we create a Python file with a filename extension `.py`
- For example, we create a file `calculater.py`. It contains the following code.

```
x = 10
y = 5
```

```
z = x * y
print(z)
```

- Then in terminal, we type `python3 calculator.py`

```
$ python3 calculator.py
50
```

- NOTE:
  - Before running `python3` command, you should make sure your current directory is where the file `calculator.py` is located. If not, use `cd` command to direct it.
  - In VS code, you can use go to `file->open folder` to open the folder that you would like to create the file or run the file. Then the terminal current directory will be exact the folder you would like to go to.

4. Values and types

(Chapter 1.5)

Integer or int

- Our first program

```
10/2
```

- The numbers here are **integer**. It is referred to as an `int`. We can use `type` function to see what type is the value or variable.

```
>>> type(10)
<class 'int'>
```

- For a variable,

```
>>> n = 388
>>> type(n)
<class 'int'>
```

Float

- Floating-point numbers, e.g., `10.0`, belong to `float`.

```
>>> type(10.0)
<class 'float'>>
```

- For a variable

```
>>> n = 3.14
>>> type(n)
<class 'float'>
```

String

- String value is the value with quotation marks.

```
>>> type('Hello')
<class 'str'>
```

- or

```
>>> message = 'Hello World!'
>>> type(message)
<class 'str'>
```

5. Strings

Values in Quotation marks

- Actually, we can either use double quotation marks or single quotation marks.

```
>>> s = "hello"
>>> s
'hello'

>>> s = 'hello'
>>> s
'hello'
```

Small problem with quotation marks

- What if one of the character in the string is quotation mark?

```
>>> s = ""Hello!", he said."
  File "<stdin>", line 1
```

```
    s= ""Hello!", he said."
         ^
SyntaxError: invalid syntax
```

- There is an error, an invalid syntax. We cannot include *doubt quotation marks* in *doubt quotation marks*.
- Solutions

```
>>> s = "'Hello!', he said."
>>> s
"'Hello!', he said."

>>> s = '"Hello!", he said.'
>>> s
'"Hello!", he said.'

>>> s = "\"Hello!\", he said."
>>> s
'"Hello!", he said.'
```

String concatenation

- The + operator performs string concatenation, which means it joins the strings by linking them end-to-end. For example:

```
>>> s = 'Hello, '
>>> name = 'Python!'
>>> s + name
'Hello, Python!'
```

String Repetition

- The * operator also works on strings; it performs repetition. For example,

```
>>> s = 'Hello'
>>> s * 3
'HelloHelloHello'
```

String Built-in methods

- String Built-in methods: https://docs.python.org/3/library/stdtypes.html#string-methods

6. String formatting and f-string

- If we run the following program.

```
name = 'Jin'
print('Hello, ' + name)
```

- In this program, we use + operator to concatenate two strings. The output is

```
Hello, Jin
```

- It is good. But if we would like to print a string together with an integer, the concatenation operation doesn't work. For example,

```
n = 35
print("The number of students is " + n)
```

- It generates the following error, which says that Python can only concatenate string (not "int") to string.

```
    print("The number of students is " + n)
          ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~
TypeError: can only concatenate str (not "int") to str
```

- We may convert int to string by using `str()` function first, then do concatenation, like this

```
n = 35
n_as_str = str(n)
print("The number of students is " + n_as_str)
```

- Seems good. However, this kind of conversion could be annoying. For example, if we would like to print a sentence with more number of int.

```
x = 3
y = 4

z = x + y

# convert x, y, z to str
x_as_str = str(x)
y_as_str = str(y)
z_as_str = str(z)

# print
print(x_as_str + " plus " + y_as_str + " equals " + z_as_str)
```

- The above program works, but it is really not elegant. First, it is boring to convert all the integers to strings. Second, there are so many + symbols that we cannot clearly see what we are printing.
- Fortunately, in Python 3.6 and above, it gives as the *f-strings*, which make the string formatting much easier. We can do it like this.

```
x = 3
y = 4

z = x + y

print(f"{x} plus {y} equals {z}.")
```

7. Create our own function by `def`

3.1 First function without arguments

- Recall our code.

```
name = "Jin"
print("Hello, ", end="")
print(name)
```

- Now, suppose we have more than one students, and need to say hello to all students. We can do this

```
name_1 = "Jin"
print("Hello, ", end="")
print(name_1)

name_2 = "Mark"
print("Hello, ", end="")
print(name_2)

name_3 = "Nancy"
print("Hello, ", end="")
print(name_3)
```

- Output:

```
Hello, Jin
Hello, Mark
Hello, Nancy
```

- In this code, give different values to variable `name`. It will print different name, while it always prints `Hello, ` . We type `print("Hello, ", end="")` repeatedly. So we may create a function such that we can reuse the action. ```python def hello(): print("Hello, ", end="")

```
name_1 = "Jin"
hello()
print(name_1)

name_2 = "Mark"
hello()
print(name_2)

name_3 = "Nancy"
hello()
print(name_3)
```

- Notice that everything under `def hello()` is **indented**. Python is an indented language. Indentation or four spaces.
- Python is going to treat lines of code that I indent underneath this line as the meaning of this new function.

3.2 Function with Arguments

- We can further improve the program since the structure of the last three blocks are similar, that is, print "hello, ", then print name. So we may combine the two lines.

```
def hello(nm):
    print("Hello, ", end="")
    print(nm)

name_1 = "Jin"
hello(name_1)

name_2 = "Mark"
hello(name_2)

name_3 = "Nancy"
hello(name_3)
```

- Actually, we still can improve the program further after we learn more. We will discuss it later.

3.3 Function with default value

- Let us discuss the greeting word `Hello`. Sometimes, we may use different words, e.g., Hi, Hey, Good Morning, et. al.

- Suppose we would like to say Hello to Jin and Mark, but Morning to Nancy. How can we revise the code above .
- Let's create the Pseudocode.

```
# create hello() function


name_1 = "Jin"
# say "Hello" to Jin
hello(name_1)

name_2 = "Mark"
# say "Hello" to Mark
hello(name_2)

name_3 = "Nancy"
# say "Morning" to Nancy
hello(name_3)
```

- Then finish the function

```
# create hello() function
def hello(nm, greeting="Hello "):
    print(greeting, end="")
    print(nm)

name_1 = "Jin"
# say "Hello" to Jin
hello(name_1)

name_2 = "Mark"
# say "Hello" to Mark
hello(name_2)

name_3 = "Nancy"
# say "Morning" to Nancy
hello(name_3, "Morning ")
```

- **NOTE:** *Default argument must follow non-default argument.*

8. Return values

- Let us create a useful function `add()`, which return a summation of two numbers.

```
def add(a, b):
    c = a + b
    return c
```

```
x = 10
y = 5

print(add(x, y))
```

- `x` and `y` are passed to `add()` function. Then the calculation of `x+y` is returned back.

9. Variables and parameters are local

- Let us see this code:

```
def add(a, b):
    c = a + b
    return c

x = 10
y = 5

print(add(x, y))

print(x)
print(c) # will get an error
```

- You will get an error

```
    print(c)
          ^
NameError: name 'c' is not defined
```

- Because *when you create a variable inside a function, it is local, which means that it only exists inside the function.*

10. Organizing our code by using `main()`

- Let us see the code above, we create the `add()` function, then we may call or use it again and again. We don't want to see it, but just want to use it. Hence, *We don’t like to have our functions at the start of our program.*
- So we may try this

```
x = 10
y = 5

print(add(x, y))

def add(a, b):
```

```
    c = a + b
    return c
```

- However, we will get an error

```
    print(add(x, y))
          ^^^
NameError: name 'add' is not defined
```

- seems that the compiler doesn't know what is `add()`! This is because we cannot use it before the function is created.
- We need to tell the compiler that we have *a `main` function* and we have a separate `hello` function.

```
def main():
    x = 10
    y = 5

    print(add(x, y))

def add(a, b):
    c = a + b
    return c

main()
```

- In the code above, we *create a `main` function (means the main part of our code)*, in which `add()` function is called. However, we didn't use either `main()` or `add()` until Line 11.

Practice Questions

1. Use `input` function to give show a prompt to

    - tell the user the program can return the product of two numbers, and
    - Let users input two numbers
    - At last, return the result.

    Following is the expected outcome.

```
You can input two numbers. I will return the product the two numbers.
Please input the first number: 2.3
Please input the second number: 3.4
2.3 times 3.4 equals 7.819999999999999.
```

2. Write a program to create a function `show_employee()` using the following conditions.

- It should accept the employee's name and salary, and print both.
- If the salary is missing in the function call, then assign default value 9000 to salary.

Given

```
showEmployee("Ben", 12000)
showEmployee("Jack")
```

Expected output:

```
Name: Ben, salary: 12000

Name: Jack, salary: 9000
```

3. Calculate the area of a circle. The mathematical formula is πr^2, where r denotes the radius, and π is the mathematical constant, 3.14.

    1. create a function named `square`, which returns the squared value given a number.

    2. create a function named `area`, which returns the area of a circle given a radius. In this function, you need to call the `square` function defined in (1).

    3. Create a main function to orgnize the program. In the main function, print a sentence like

        - "The area of a circle with radius 1 is 3.14." if you put $r=1$, or
        - "The area of a circle with radius 2 is 12.56." if you put $r=2$.

Summing up

- Arithmetic operators
- Variables
- Terminal
- `.py` file
- Types, int, float, and string
- String concatenation and repetition
- f-strings
- Create function using `def`
- Arguments, default value
- Local variables
- `main()` function